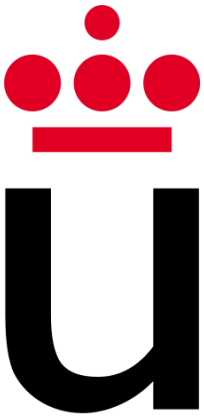

Universidad Rey Juan Carlos

Escuela Técnica Superior en Ingeniería Informática

Grado en Ingeniería Informática

Curso Académico 2020/2021

Trabajo de Fin de Grado

Creación del videojuego educativo “L.U.C.A.” para la enseñanza de la evolución de las especies realizado con Unity

Autor del proyecto:

Juan González Puerta

Tutora del proyecto:

Raquel Belén Hijón Neira

RESUMEN

En este Trabajo de Fin de Grado se ha desarrollado un videojuego educativo para ordenador con Unity sobre la evolución biológica.

La aplicación se llama L.U.C.A. y consiste en un juego de estrategia en el que se debe evolucionar una especie para que sobreviva en un mapa que contiene:

- Biomas: un total de 13 biomas diferentes repartidos por el mapa en base a la temperatura.
- Otras especies: el jugador debe enfrentarse a especies con diferentes adaptaciones.
- Catástrofes: desastres naturales que ocurren de forma periódica y modifican el mapa completamente.

Comenzando por el último antepasado común universal (conocido en inglés como L.U.C.A. Last Universal Common Ancestor), el jugador debe expandir su especie por un mapa con diferentes biomas. Si no está suficientemente adaptado al bioma, no podrá expandirse a esa zona. A medida que avanza la partida, se consiguen puntos que se podrán cambiar por rasgos evolutivos. Los rasgos permiten adaptarse a nuevos biomas, tener un tamaño mayor o reproducirse más rápido.

A medida que el juego va avanzando, el jugador irá descubriendo nuevas partes del mapa y se topará con otras especies que pueden estar mejor adaptadas que la del jugador. Esto añade un componente estratégico ya que el jugador deberá decidir qué medio es el mejor para colonizar y expandirse.

Por último, cada cierto tiempo ocurrirá una catástrofe que afectará a todo el mapa, tanto a los biomas como a las especies. Una de las catástrofes puede ser una bajada de temperaturas, lo que provoca que los climas fríos se vuelvan gélidos, y climas cálidos se vuelvan templados, además de que se congelen los océanos u otras masas de agua del mapa.

Después de desarrollar el videojuego, se han realizado unas encuestas para comprobar los conceptos que los jugadores pueden aprender tras haber jugado una cantidad de tiempo considerable (una encuesta antes de jugar, para comprobar los conceptos que los jugadores tienen sobre la evolución biológica, y otra después para comprobar los conceptos aprendidos).

La aplicación se encuentra alojada en el siguiente enlace y se puede jugar tanto en inglés como en español: https://sites.google.com/view/lucaelvideojuego/inicio

SUMMARY

In this Final Degree Project, I've developed an educational video game for computers with Unity about biological evolution.

The app is called L.U.C.A. (Last Universal Common Ancestor) and consists of a strategy game in which the player must evolve a specie to survive on a map replete with:

- Biomes: a total of 13 different biomes spread over the map based on their temperature.
- Other species: the player faces different species with different adaptations.
- Catastrophes: natural disasters that occur periodically that completely modify the map.

Starting from the Last Universal Common Ancestor (L.U.C.A.), the player must expand the species through a map full of different biomes. If the specie is not enough adapted to that biome, it will not be able to expand to that area. As the game progresses, the player earns points that can be exchanged for evolutionary traits. These traits will allow you to adapt to new biomes, have a larger size or reproduce faster.

As the game progresses, the player will discover new parts of the map and will come across other species that may be better adapted than the player´s one. This adds a strategic component since you have to decide which medium is the best to colonize and expand.

Finally, from time to time a catastrophe will occur that will affect the entire map, both biomes and species. One of the catastrophes can be Global cooling, which causes cold climates to become icy, and hot climates become temperate, and the freezing of oceans or other areas with water present in the map.

After the development of the video game, surveys have been conducted to verify the concepts that players can learn after having played a considerable amount of time (a first survey before playing it to validate the concepts players have about biological evolution, and a second survey afterwards to check the concepts learned).

The application is hosted at the following link and can be played in both English and Spanish: https://sites.google.com/view/lucaelvideojuego/inicio

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

En este documento se describe el desarrollo del trabajo de fin de grado, incluyendo la investigación y documentación previa, el desarrollo del proyecto y los resultados obtenidos.

1.1 Descripción

El objetivo de este proyecto es realizar un videojuego educativo para ordenador sobre la evolución biológica de las especies.

En este juego, la tarea de los usuarios es controlar una especie llamada LUCA, en un mapa con diferentes biomas, especies a las que enfrentarse y cambios medioambientales. El objetivo del juego es sobrevivir el mayor tiempo posible, adaptándose a las diferentes situaciones.

En cada partida, el jugador deberá ir evolucionando su especie con diferentes rasgos evolutivos, seleccionándolos cuidadosamente si quiere que su especie no se extinga. A continuación, en la Figura 1 se puede ver una captura de la versión terminada del juego.

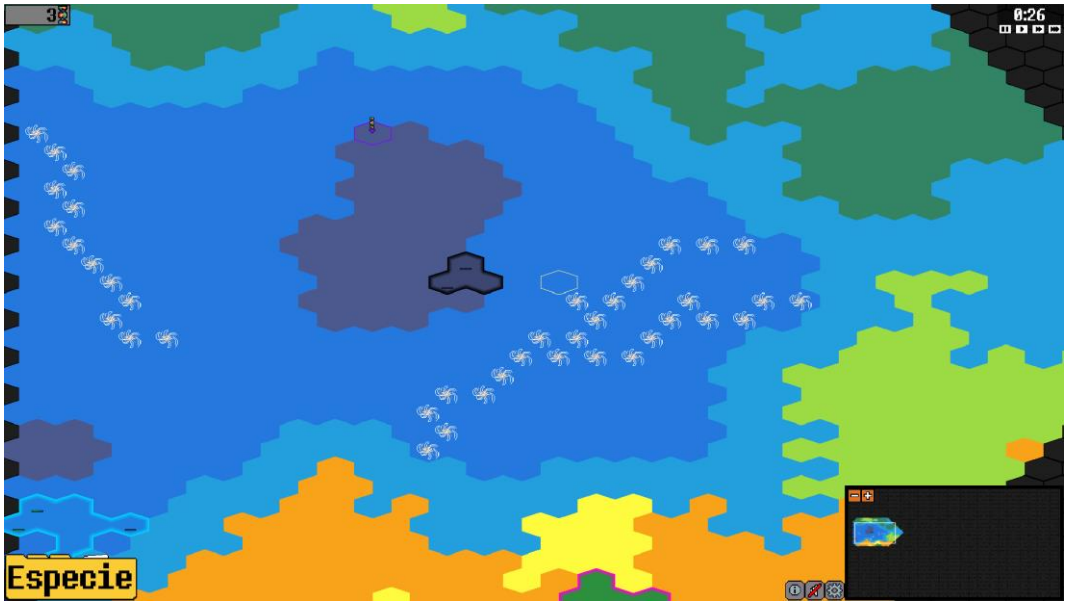

Figura 1: Versión final del juego.

El público objetivo se constituye principalmente por personas con bajo conocimiento de la evolución biológica. Dentro de este público, se ha orientado hacia estudiantes de instituto ya que están aprendiendo conceptos similares a los explicados en el juego.

1.2 Motivación

La paleontología siempre ha sido una de mis pasiones desde que era un niño. El hecho de pensar que hace millones de años vivían seres completamente diferentes a los que existen ahora me parecía fascinante. Por otro lado, otra de mis pasiones está relacionada con el mundo de la enseñanza, ya que llevo siendo monitor de ocio en una asociación scout más de 4 años.

Juntando dos de mis grandes pasiones y mis conocimientos sobre informática, pensé en crear un videojuego en el que se enseñaran los fundamentos de la evolución de una forma dinámica y entretenida.

Si bien es cierto que dentro de la evolución biológica se pueden tratar temas muy avanzados, lo fundamental es tener una buena base de conceptos básicos. Por ello, la intención de este proyecto es transmitir conceptos básicos, claros y sencillos sobre la evolución orientado a un público con pocos conocimientos sobre la evolución biológica.

Al analizar el mercado actual de videojuegos que tratan la evolución biológica como tema principal o transversal, he descubierto que la mayoría tratan temas muy específicos por encima, sin llegar a profundizar y esto puede llevar a confusiones con conceptos.

2. ESTADO DEL ARTE

En este apartado de la memoria, se analizan los distintos recursos disponibles para la realización de este proyecto.

2.1 Videojuegos y la evolución

La mayoría de los videojuegos con una temática basada en la evolución biológica se pueden dividir en dos grandes grupos: los simuladores de evolución automática y los juegos en los que se evoluciona de forma activa.

En el primer grupo, englobamos a los juegos que requieren de muy poca o nula selección de parámetros de una especie al principio de la partida, para luego simular un entorno y ver cómo evoluciona sin la intervención del jugador.

En el segundo grupo, englobamos a los juegos en los que se controla una especie en concreto, que el jugador deberá de adaptar con mejoras al medio en el que se encuentra.

2.2 Simuladores de evolución automática

Como se ha indicado anteriormente, en este grupo incluimos juegos en los que el jugador es espectador y presencia los cambios y adaptaciones de las especies en un ecosistema cambiante.

A continuación, se analizan diversos juegos que tengan como temática la evolución biológica y cumplan con estos criterios.

2.2.1 SimLife – Maxis (1992)

SimLife es un videojuego estilo sandbox en el que el jugador debe simular un ecosistema autosuficiente.

El jugador puede añadir diferentes tipos animales y plantas a un mundo generado aleatoriamente o customizado por el jugador. Tanto los animales como las plantas se pueden modificar genéticamente para comprobar si estas nuevas especies pueden sobrevivir en diferentes entornos (Figura 2).

El objetivo del juego es experimentar la interacción de las diferentes especies en diferentes biomas, y ver cómo sería un ecosistema funcional y estable, y qué especies hacen falta para llevarlo a cabo.

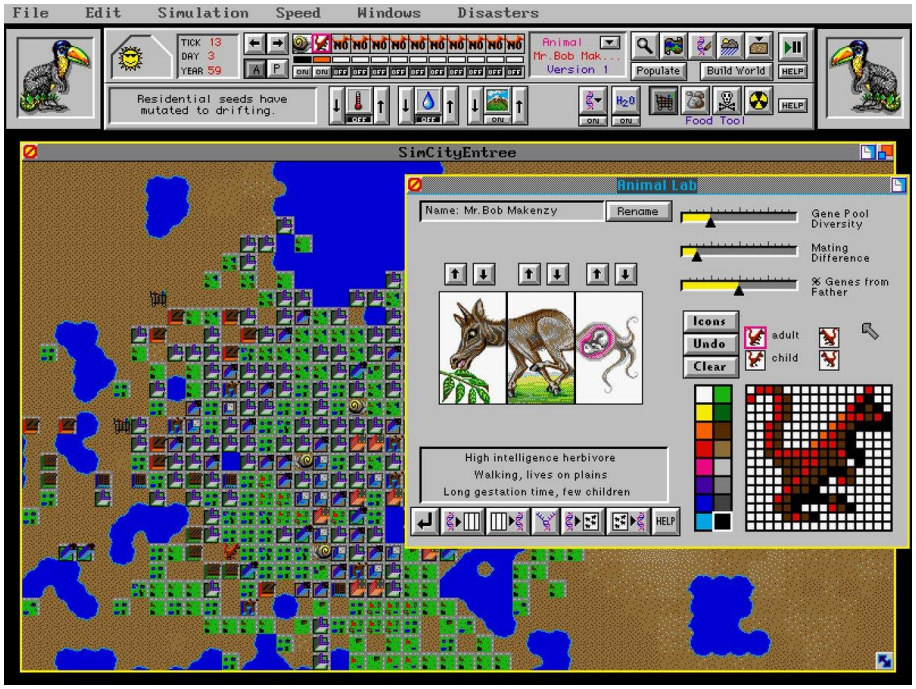

Figura 2: SimLife.

2.2.2 Selection Game – Adventure Club Games (2011)

Selection Game es un videojuego de simulación en el que unos escarabajos atraviesan diferentes biomas.

El jugador elije los diferentes biomas por los que los escarabajos tendrán que pasar. Seguidamente comienza la carrera y los primeros que lleguen se reproducirán y pasarán sus rasgos a sus descendientes. (Figura 3).

El objetivo del juego es enseñar varios conceptos: dentro de una misma especie puede haber variaciones dependiendo del bioma en el que se encuentren (alas más grandes, antenas más pequeñas, etc), los rasgos son heredables y se heredan de las especies mejor adaptadas a los biomas.

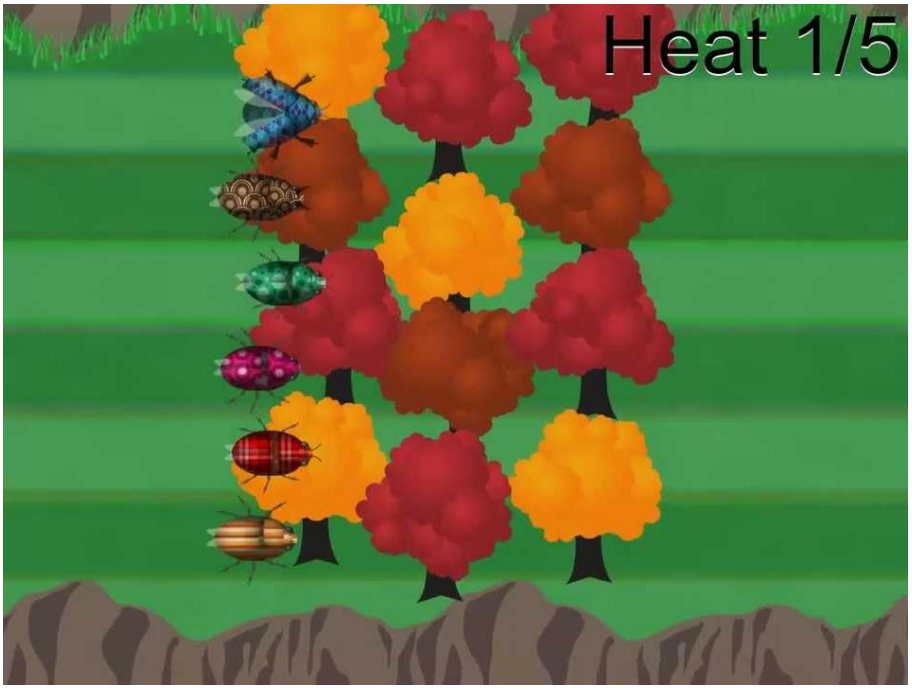

Figura 3: Selection Game.

2.2.3 Cell to Singularity – Computer Lunch (2018)

Cell to Singularity es un videojuego estilo clicker en el que se desbloquean diferentes rasgos/evoluciones a medida que ganas puntos.

Empezando desde moléculas simples como aminoácidos, se eligen rasgos y desbloqueas un mapa que contiene diferentes rasgos evolutivos de los seres vivos (este juego también incluye la evolución de la civilización, pero lo he incluido por su gran biblioteca de evoluciones biológicas) (Figura 4).

El objetivo del juego es desbloquear el mapa completo de evoluciones para conseguir la mayor puntuación posible. También tiene una variante específica dedicada al periodo mesozoico (la era de los dinosaurios).

Figura 4: Cell to Singularity.

2.2.4 Species: Artificial Life – Quasar (2018)

Species: Artificial Life es un videojuego de simulación 3D que replica la selección natural en tiempo real.

En este juego se controlan diferentes especies dentro de un entorno y se experimenta una selección natural realizada por una inteligencia artificial, la cual hace evolucionar a las diferentes especies dependiendo de las variaciones que introduzca el jugador (Figura 5).

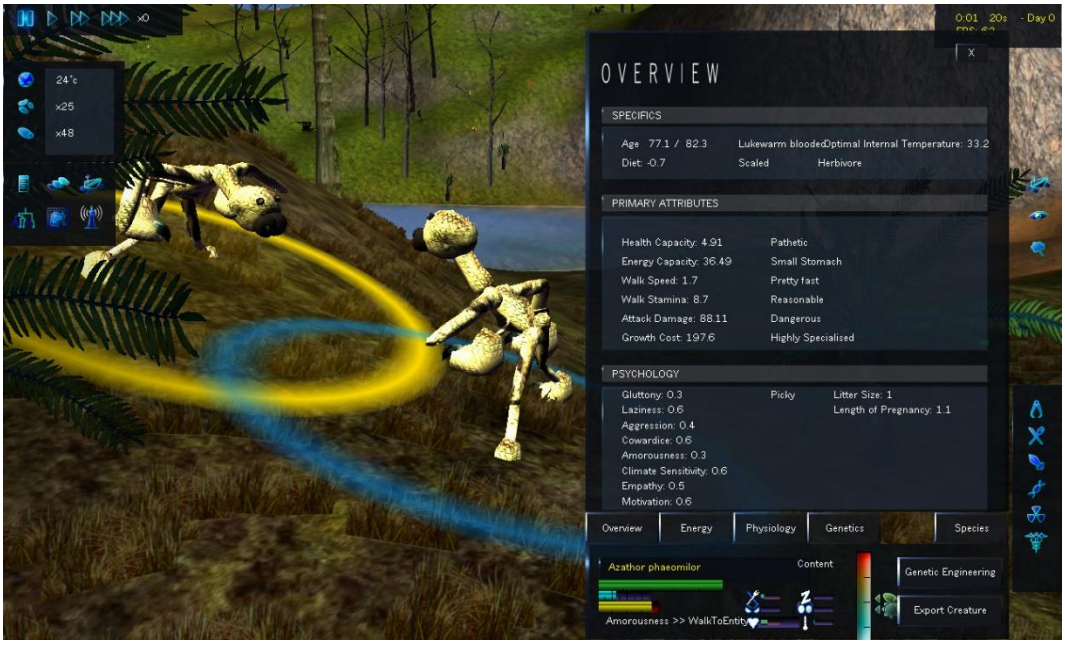

Figura 5: Species: Artificial Life.

2.3 Juegos de evolución activa

Como se ha indicado anteriormente, en este grupo se incluyen juegos en los que el jugador debe seleccionar los rasgos de la especie que va a evolucionar, por lo que es más activo en el gameplay.

A continuación, se analizan diversos juegos que tienen como temática la evolución biológica y cumplen estos criterios.

2.3.1 Spore – Maxis (2008)

Spore es un videojuego de simulación de vida y estrategia que simula la evolución de una especie desde las etapas más primitivas (seres unicelulares), hasta la colonización de la galaxia.

La especie del jugador pasa por diferentes etapas definidas por el juego: etapa de célula, de criatura, de tribu, de civilización y de galaxia. En cada etapa puede mejorar diferentes aspectos de la especie que controla mediante un sistema de puntos. (Figura 6).

Figura 6: Spore.

2.3.2 Sparkle 2 Evo – Forever Entertainment S. A. (2011)

Sparkle 2 Evo es un videojuego en el que la especie comienza siendo un pequeño organismo y debe alimentarse para aumentar de tamaño.

El medio en el que se desarrolla este juego es acuoso y la especie debe buscar nutrientes dentro del mapa que se asemeja a un abismo. Llegado un punto la especie es tan grande que puede consumir otros organismos. El objetivo es hacerse tan grande como pueda. (Figura 7).

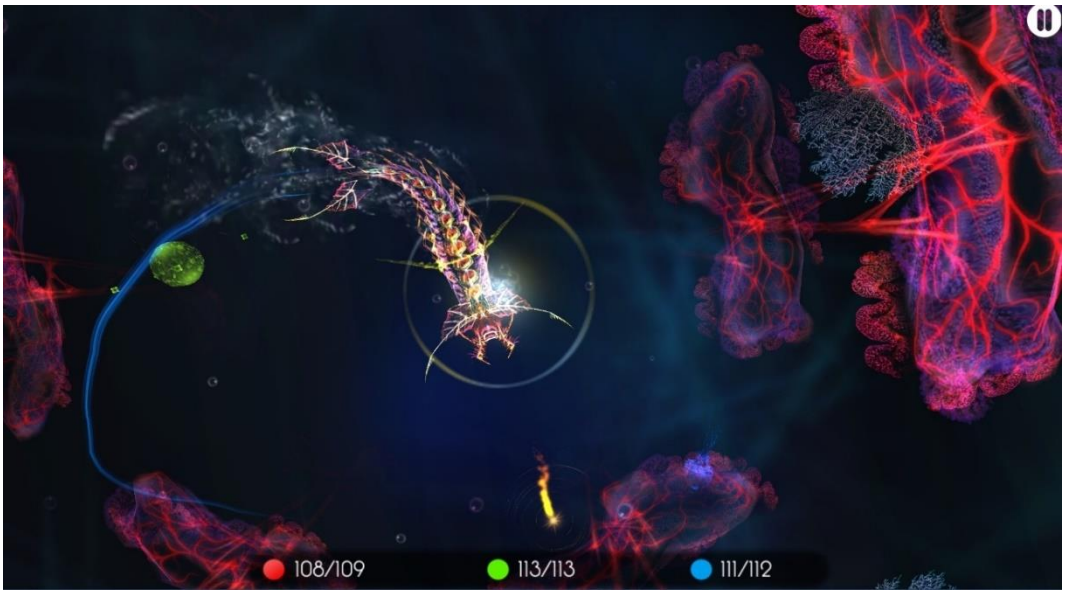

Figura 7: Sparkle 2 Evo.

2.3.3 Niche – Stray Fawn Studio (2016)

Niche es un videojuego de estrategia que en el que se tiene que garantizar la supervivencia de diferentes individuos de la especie del jugador.

En este juego de estrategia a base de turnos con elementos "rogue", se debe desarrollar una especie propia basada en rasgos genéticos reales. A lo largo de la partida hay que enfrentarse a depredadores, cambios climáticos, propagación de enfermedades… (Figura 8).

Figura 8: Niche.

2.3.4 Evolution Board Game – North Star Digital Studios (2019)

Evolution Board Game es un videojuego de cartas inspirado en un juego de mesa del mismo nombre, en el que varias especies de distintos jugadores compiten por los escasos recursos.

La partida consta de varios jugadores que crean y modifican sus especies con las cartas conseguidas. Las especies pueden ser herbívoras, carnívoras u omnívoras y en cada turno deben de alimentarse dependiendo de su dieta. El objetivo del juego es conseguir la mayor cantidad de alimentos posible para conseguir una puntuación mayor. (Figura 9).

Figura 9: Evolution Board Game.

2.4 Tabla de Conclusiones

En la siguiente tabla (Tabla 1) se han comparado los juegos analizados en este trabajo teniendo en cuenta su temática, mecánica y accesibilidad.

Juego	Teoría Evolutiva	Estrategia	Biomas	Catástrofes	Múltiples Especies	Explicación de Rasgos	Disponible en español
SimLife	Sí	No	Sí	Sí	Sí	Sí	No
Selection Game	Sí	No	Sí	No	No	Sí	No
Cell to Singularity	Sí	No	No	No	No	Sí	Sí
Species: Artificial Life	Sí	Sí	Sí	No	Sí	No	No
Spore	No	Sí	No	No	Sí	No	Sí

Sparkle 2 Evo	No	Sí	No	No	Sí	No	No
Niche	Sí	Sí	Sí	Sí	Sí	No	Sí
Evolution Board Game	Sí	Sí	No	No	Sí	No	No

Tabla 1: Comparación de videojuegos similares al proyecto

Entre los videojuegos abalizados no se ha encontrado ninguno en que se simule la expansión de territorios de las especies con un modelo adaptativo flexible. "L.U.C.A" mejora algunas de las carencias que tienen los juegos anteriores para crear un juego en el que se explican los fundamentos de la evolución.

Además, la mayoría de los juegos tratan la teoría evolutiva sin explicar la importancia de los rasgos que posee la especie. La falta de esta información dificulta un entendimiento más avanzado sobre la evolución. Por lo anterior, los rasgos evolutivos de las especies son una característica importante de este juego.

3. HERRAMIENTAS DEL DESARROLLO

En esta sección se explican las herramientas empleadas en el desarrollo de esta aplicación, así como para qué han sido utilizadas en el juego.

3.1 Hardware

3.1.1 Ordenador de sobremesa

La herramienta fundamental en el desarrollo de un proyecto de este tipo es un ordenador adecuado. Para esta aplicación se ha utilizado un ordenador de sobremesa con las siguientes especificaciones:

- Sistema Operativo: Windows 10
- Procesador: AMD Ryzen 7 1700
- Tarjeta Gráfica: NVIDIA GeForce GTX 1060
- Memoria RAM: 16,00 GB

3.2 Software

3.2.1 Unity 2018.4.30f1

El juego se ha desarrollado en el motor de Unity, ya que ofrece de forma gratuita todo lo necesario para un proyecto de este tipo, con la posibilidad de distribuir la aplicación en otras plataformas.

3.2.2 Visual Studio 2019

Para la programación del videojuego se ha utilizado el editor de código Visual Studio que, además de ser gratuito para estudiantes, ofrece herramientas de desarrollo y depuración para Unity.

3.2.3 Aseprite

Los diseños de los elementos visuales de la aplicación se han realizado en Aseprite, ya que es uno de los mejores programas para el estilo de dibujo pixelart.

3.2.4 Google Forms

Las encuestas que han rellenado los usuarios se han elaborado en Google Forms. Es un software con facilidad de uso, análisis de datos y la mayoría de los usuarios están acostumbrados a utilizarlo.

4. Descripción Informática

Debajo se detallan los diferentes aspectos técnicos en los que se ha basado el desarrollo del proyecto.

4.1 ANÁLISIS

A continuación, se expone el desarrollo técnico de la aplicación, concretamente los requisitos funcionales y no funcionales junto con los casos de uso.

4.1.1 Objetivos

Una vez concretada la idea, es necesario definir una serie de objetivos para la aplicación. Al ser la enseñanza de la teoría evolutiva la meta principal del proyecto es primordial explicar la funcionalidad de cada rasgo evolutivo. Igualmente, para enfocar el juego en la enseñanza de la evolución y no en la mecánica del juego, es importante crear una interfaz intuitiva y simple.

Los conceptos que se trabajan durante el juego son los siguientes:

- **Biomas**: enseñar los principales biomas que existen actualmente y sus diferencias.
- **Rasgos evolutivos**: definir qué es un rasgo evolutivo y su funcionalidad.
- **Selección natural**: explicar transversalmente la selección natural mediante la selección de los organismos mejor adaptados.
- **Catástrofes naturales**: mostrar que también la selección natural tiene un factor de suerte.

A continuación, se enumeran los objetivos del proyecto junto con una breve explicación:

- OBJ-01 Explicar la teoría evolutiva: describir de una forma sencilla los fundamentos de la teoría evolutiva.
- OBJ-02 Enseñar la relación entre rasgos y biomas: enseñar cómo están relacionados los rasgos evolutivos de diferentes animales con los biomas.
- OBJ-03 Enseñar las consecuencias de las catástrofes: enseñar cómo afecta una catástrofe natural a diferentes biomas y especies.
- OBJ-04 Menú principal: crear una pantalla que permita acceder al resto.
- OBJ-05 Introducir rasgos evolutivos: introducir rasgos evolutivos en el juego con sus respectivas descripciones.
- OBJ-06 Especies secundarias: implementar especies diferentes a las del jugador dentro del juego.
- OBJ-07 Mapas aleatorios: implementar un generador de mapas orgánicos con diferentes biomas.
- OBJ-08 Diseño gráfico: diseñar los elementos visuales del juego con un estilo pixelart.

4.1.2 Requisitos Funcionales

Los requisitos funcionales definen las funciones y servicios que un sistema debe desempeñar. Estas funciones y servicios se describen como el conjunto de entradas, comportamientos y salidas que puede presentar dicho sistema (Tabla 2).

Nombre	Descripción
Instalar la aplicación	El usuario será capaz de poder instalar la aplicación de una forma fácil y sencilla.
Iniciar la aplicación	El usuario será capaz de poder iniciar la aplicación con un ejecutable.
Instrucciones	Al principio de la partida el sistema mostrará unas instrucciones que describan cómo jugar.
Cambio de idioma	El usuario será capaz de cambiar el idioma del juego (Inglés y Español).
Navegación	El sistema permitirá al usuario navegar a través de diferentes pantallas de la aplicación de una forma sencilla.
Empezar partida	El usuario será capaz de comenzar una partida sin problemas.
Instrucciones de juego	El sistema mostrará las instrucciones de juego al inicio de la partida, y estarán accesibles para posibles consultas.
Feedback	El sistema mostrará de forma clara las consecuencias de las acciones que realiza el usuario.
Cerrar la aplicación	El usuario será capaz de cerrar la aplicación desde cualquier lugar dentro de la aplicación.

Tabla 2: Requisitos funcionales

4.1.3 Requisitos no Funcionales

Los requisitos no funcionales (Tabla 3) son los que hacen referencia a los aspectos a cumplir por elementos externos a la aplicación.

Nombre	Descripción

Sistema operativo	Como mínimo Windows 7, MAC OS 10.6 o Linux.
Interfaz gráfica	El sistema debe de proporcionar una interfaz gráfica fácil y sencilla de comprender por el usuario.
Usabilidad	El sistema debe tener una dificultad de uso baja.
Espacio en disco	El usuario deberá tener un sistema con un mínimo de espacio disponible de 77 MB.

Tabla 3: Requisitos no funcionales

4.1.4 Casos de Uso

Los casos de uso se utilizan para describir acciones o actividades que tiene que realizar un actor (en este caso, el usuario) para realizar un determinado proceso (Tablas 4-9):

Nombre	Caso de Uso 1 – Hacer zoom en el mapa
Descripción	El usuario desea acercar la cámara del mapa.
Precondición	Haber iniciado la aplicación.
Flujo Normal	1. El usuario inicia una partida 2. El usuario cierra las ventanas del tutorial 3. El usuario hace click en el botón de "+"de la interfaz o mueve la rueda del ratón hacia delante.
Flujo Alternativo	La cámara está acercada el máximo posible y no avanzará.
Postcondiciones	La cámara se acercará a la posición indicada

Tabla 4: Caso de uso 1

Nombre	Caso de Uso 2 – Evolucionar un rasgo
Descripción	El usuario desea adquirir un nuevo rasgo evolutivo para su especie.
Precondición	Haber iniciado la aplicación. Haber empezado una partida.

Flujo Normal	1. El usuario accede a la carpeta de la Especie. 2. El usuario selecciona la pestaña de Evoluciones. 3. El usuario selecciona el rasgo que desea evolucionar. 4. El usuario presiona el botón de evolucionar para intercambiar los puntos que tiene por el rasgo.
Flujo Alternativo	El usuario no tiene suficientes puntos y no puede presionar el botón de evolucionar.
Postcondiciones	La especie del usuario adquiere el rasgo nuevo.

Tabla 5: Caso de uso 2

Nombre	Caso de Uso 3 – Volver al menú de inicio
Descripción	El usuario desea volver al menú de inicio.
Precondición	Haber iniciado la aplicación. Haber empezado una partida.
Flujo Normal	1. El usuario accede al menú de pausa mediante el icono de la interfaz o presionando la tecla de escape. 2. El usuario confirma que quiere volver al menú de inicio.
Flujo Alternativo	
Postcondiciones	Se regresa al menú de inicio.

Tabla 6: Caso de uso 3

Nombre	Caso de Uso 4 – Ver las características de otra especie
Descripción	El usuario desea ver el tamaño que tiene una especie.
Precondición	Haber iniciado la aplicación. Haber empezado una partida.

Flujo Normal	1. El usuario selecciona una casilla en la que se encuentra la especie. 2. El sistema muestra la ventana de información de la casilla seleccionada. 3. El usuario presiona el botón con el nombre de la especie de la ventana. 4. El sistema muestra la ventana de características de la especie.
Flujo Alternativo	El usuario intenta ver las características de su propia especie, esto se debe hacer accediendo a la carpeta de Especie.
Postcondiciones	Se visualizan las características de la especie.

Tabla 7: Caso de uso 4

Nombre	Caso de Uso 5 – Ver la temperatura de un bioma
Descripción	El usuario desea visualizar la cantidad de temperatura que tiene un bioma.
Precondición	Haber iniciado la aplicación. Haber empezado una partida.
Flujo Normal	1. El usuario selecciona una casilla con el bioma que quiere comprobar. 2. El usuario posiciona su ratón encima del icono “i” del menú desplegado.
Flujo Alternativo	
Postcondiciones	El jugador visualiza qué temperatura tiene el bioma que ha seleccionado.

Tabla 8: Caso de uso 5

Nombre	Caso de Uso 6 – Visualizar población total
Descripción	El usuario desea ver la población total de su especie después de una catástrofe.
Precondición	Haber iniciado la aplicación. Haber empezado una partida.
Flujo Normal	1. El usuario accede a la carpeta de la Especie.

	2. El usuario selecciona la pestaña de Población. 3. El usuario posiciona el ratón en el espacio seguido de la catástrofe que desea comprobar.
Flujo Alternativo	
Postcondiciones	El usuario visualiza la población total del momento deseado.

Tabla 9: Caso de uso 6

4.2 DISEÑO

En este capítulo se van a explicar las decisiones de diseño y usabilidad del juego, analizando cada una de las pantallas o menús que los componen. La aplicación tiene una estética estilo pixelart, que se mantiene fiel a lo largo de todo el juego.

4.2.1 Interfaz

El juego está compuesto por una interfaz que utiliza colores similares para que sea más intuitiva. También se hace uso de diferentes iconos que el usuario ha utilizado en otras aplicaciones de este estilo. Está estructurada de tal forma que la interfaz se ajusta al tamaño de la pantalla de juego. La fuente utilizada en la mayor parte del proyecto ha sido "Undertale".

4.2.1.1 Menú principal

Cuando el usuario ejecute la aplicación, la primera pantalla que se mostrará será la del menú principal (Figura 10). En la parte superior aparece el título del juego, y debajo lo que significan cada una de las siglas de las que se compone el título. Como fondo de pantalla, aparece un bosque realizado con pixelart. Además, está compuesta por 4 botones: el botón de "jugar" para iniciar el juego, el botón de "salir" para cerrar la aplicación, y dos botones para elegir el idioma inglés o español.

Figura 10: Menú principal.

4.2.1.2 Interfaz de juego

La interfaz que visualiza el jugador dentro del juego (Figura 11) está compuesta por varios elementos:

1. En la esquina superior izquierda se encuentra un contador de puntos.
2. En la esquina superior derecha se encuentra un temporizador.
3. En la esquina inferior izquierda se encuentra el botón de la especie.
4. En la esquina inferior derecha se encuentra el mini mapa y algunos botones útiles.

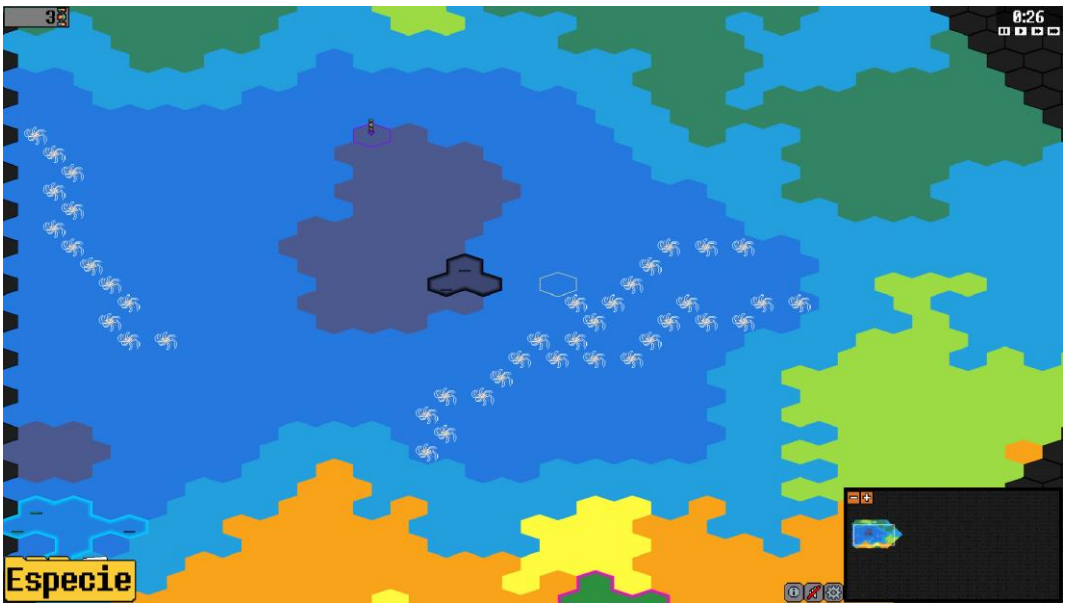

Figura 11: Interfaz de Juego.

El temporizador (Figura 12) está compuesto por un cuadro de texto que marca el tiempo que lleva jugando el jugador en un formato “mm:ss”. Debajo se posicionan 4 botones que permiten al jugador parar el tiempo, reanudarlo y aumentar la velocidad x2 y x4.

Figura 12: Temporizador.

La esquina inferior derecha (Figura 13) está compuesta por una serie de botones, el mapa y la interfaz de información de casillas.

Figura 13: Esquina inferior izquierda.

Las funciones de los botones son:

1. Con el icono de una “i”, se vuelve a mostrar el tutorial del juego.
2. Con el icono de un altavoz tachado o sin tachar, permite activar o desactivar la música del juego.
3. Con el icono de un engranaje, permite al jugador abandonar la partida volviendo al menú principal (Figura 14) (también se puede acceder con el botón de escape).

Figura 14: Ventana volver menú principal.

El mini mapa muestra el terreno que ha descubierto el jugador a lo largo de la partida. El jugador sabrá en todo momento en qué parte se encuentra gracias al rectángulo blanco que simboliza la cámara del jugador y el tamaño de la misma. A medida que el jugador visualiza otros lados del mapa (ya sea moviéndose con las teclas "wasd" o moviendo el ratón al borde de la pantalla) el rectángulo se mueve por el mini mapa. En el caso de que el jugador aumente o disminuya el zoom de la cámara (ya sea con la ruleta del ratón o con los botones "+" y "-" integrados en el mini mapa), el rectángulo también cambiará de tamaño.

La ventana de información de una casilla muestra información útil al jugador acerca de esa casilla. Primero, se muestra la imagen de la casilla que se está seleccionando, junto con el nombre del bioma que representa. También cuenta con un botón "x" para poder cerrar la ventana. En la parte superior se encuentra un icono de información (con la letra "i") que, al posicionar el ratón encima, se despliega información adicional para el jugador (Figura 15). En ésta se incluye la temperatura que tiene ese bioma, la cantidad de comida y los puntos que se ganan el expandirse a esa casilla:

Figura 15: Información adicional.

Además, si en la casilla hay una especie expandida, se pueden ver las características de dicha especie (Figura 16) haciendo click en su nombre. Al hacerlo, se desplegará una ventana que muestra el nombre de la especie, su tamaño, su expansión, su reproducción, sus rasgos principales y su adaptación a los diferentes biomas:

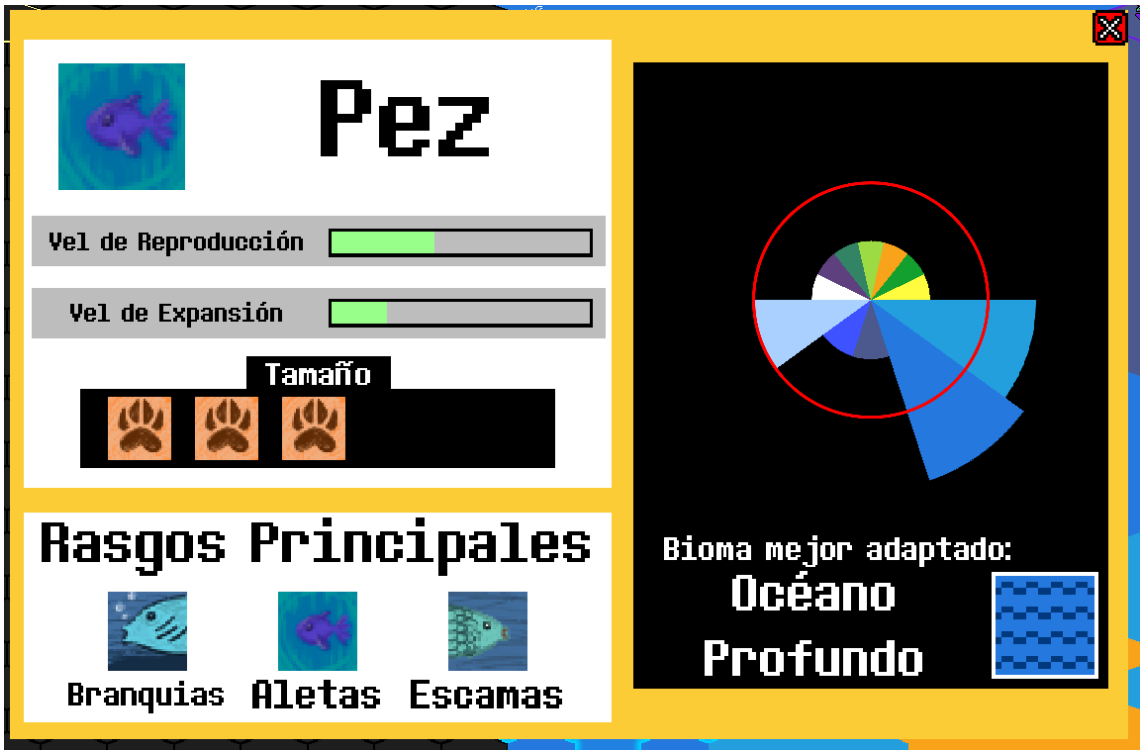

Figura 16: Características de la especie.

Cada cierto tiempo, ocurrirán catástrofes a lo largo del mapa. Estas catástrofes se anuncian por medio de una ventana (Figura 17) en la que se explica en qué consiste

la catástrofe, junto con una cuenta atrás (Figura 17) que indica cuánto tiempo queda para que se produzca dicha catástrofe:

Figura 17: Ventana de catástrofe.

En la esquina inferior derecha de la interfaz de juego existe un botón con forma de fichero (Figura 18) que nos lleva a la pantalla de la especie:

Figura 18: Botón de la especie.

Al entrar en la pantalla, se sabe cuál es el apartado por la iluminación de los botones superiores. Junto a los botones, encontramos otro contador de los puntos que tiene el jugador (igual que de la esquina superior izquierda anterior).

En la pantalla de Especie (Figura 19), el jugador puede ver las características de su especie. Primero aparece el nombre de nuestra especie (LUCA) junto a una imagen de la misma (la imagen va cambiando conforme la especie va evolucionando). Justo debajo se encuentran las barras de velocidad de reproducción y velocidad de expansión. Seguidamente, aparece el marcador de tamaño de la especie representado por un icono de garra de oso. Finalmente, en la parte inferior encontramos el bioma al que la especie está mejor adaptada. A la derecha se sitúa un gráfico circular en el que

se muestran las adaptaciones a los diferentes biomas que existen en el mapa. Cada bioma tiene un porcentaje, y sólo los que superan el 0% están por encima de la línea roja. Dicha línea indica si su especie está suficientemente adaptada a un bioma como para poder vivir en él.

Figura 19: Pantalla de Especie.

En la pantalla de Población (Figura 20) aparece un gráfico que representa la población de la especie a lo largo de la partida. Esto facilita que los jugadores relacionen cómo afectan las diferentes catástrofes a su especie, ya que éstas salen representadas en él. A medida que el tiempo avanza y el tamaño de la especie aumenta, el gráfico se va actualizando con la escala pertinente. Además, si el jugador quiere consultar un valor exacto del gráfico sólo con mover el ratón a dicho punto aparecerá un pequeño TextBox.

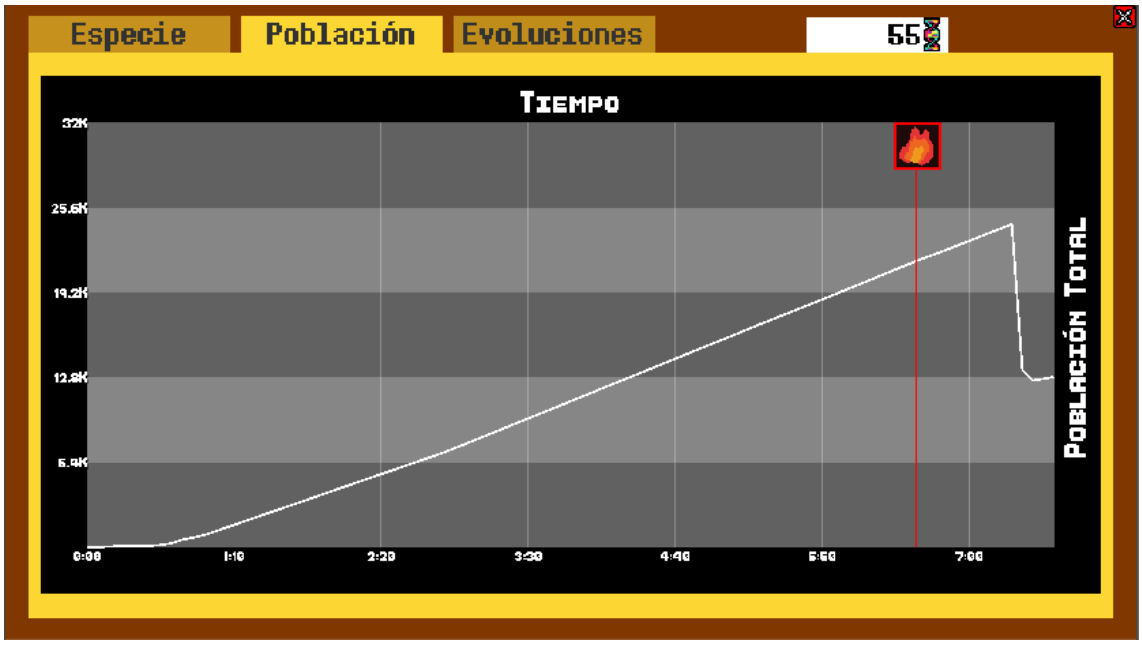

Figura 20: Pantalla de Población.

Finalmente, aparece la pantalla de Evoluciones (Figura 21). En esta pantalla el jugador puede evolucionar a su especie adquiriendo diferentes rasgos a cambio de puntos. Al hacer click en cualquiera de los rasgos, se desplegará una ventana en la que aparece el nombre del rasgo y una breve descripción biológica, además de las mejoras en los biomas, el coste y el botón para evolucionarlo. En el caso de que dicho rasgo ya se haya evolucionado, se le dará la opción al jugador de involucionar el rasgo y recuperar algunos puntos. El jugador podrá diferenciar que rasgos tiene adquiridos porque estarán más iluminados que los que no se han desbloqueado hasta ese momento.

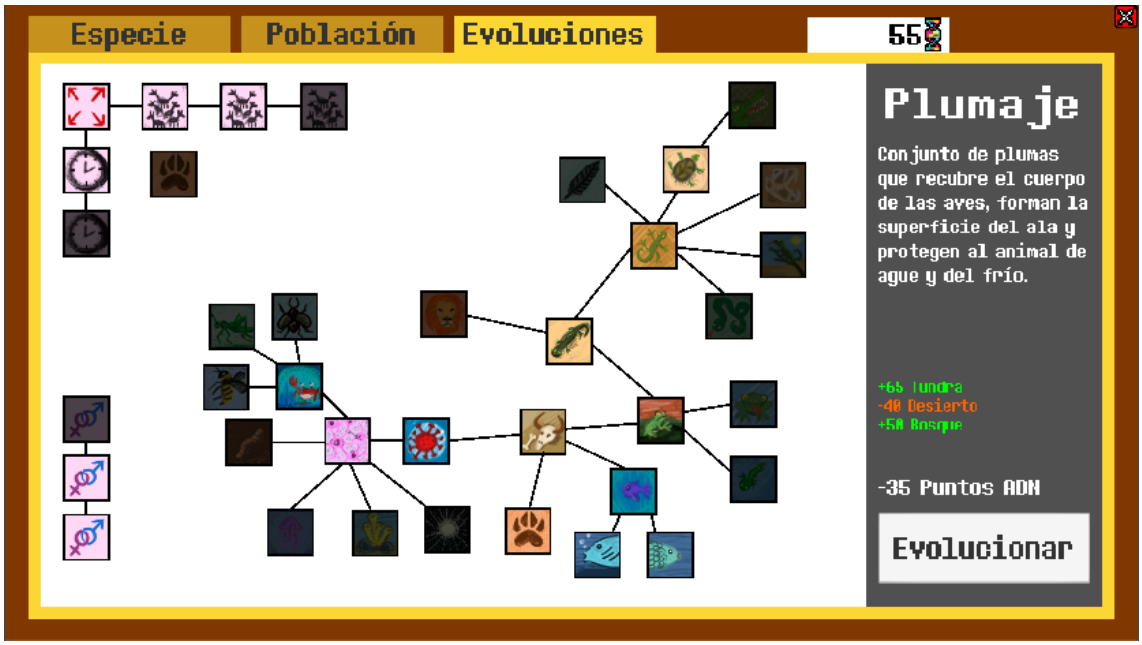

Figura 21: Pantalla de Evoluciones.

4.3 IMPLEMENTACIÓN

En este capítulo se van a explicar los aspectos más relevantes de la implementación llevada a cabo en el proyecto, tales como el funcionamiento interno que tiene, las clases principales y partes del código relevantes. El juego cuenta con dos escenas: la pantalla de inicio con el menú principal y la del juego. La escena del juego consta de un objeto y clase llamados "GameManager", cuya función principal es gestionar el funcionamiento del juego.

4.3.1 GameManager

Como se ha indicado anteriormente, la clase GameManager se encarga de cargar el juego y hacer que funcione correctamente. Existen diferentes objetos de otras clases (se explicarán con más detalle más adelante):

-**MapManager**: se encarga de mostrar y gestionar visualmente el mapa (sin incluir a las especies).

-**CellManager**: se encarga de almacenar y gestionar la información y las relaciones de todas las celdas que tiene el mapa.

-**MapGenerator**: se encarga de generar mapas orgánicos de forma aleatoria para cada partida.

-**SpeciesManager**: se encarga de gestionar las especies de la partida, tanto las secundarias como la del jugador.

-**Catastrophe**: se encarga de gestionar las catástrofes que ocurren en el juego.

-**CameraController**: se encarga de gestionar todas las funcionalidades que tiene la cámara en el juego.

Cuando el juego se inicia, se ejecuta la función de "Start" (Figura 22) del GameManager, y se realizan diferentes procesos para que sea posible empezar a jugar. Primero, se genera el mapa con la información de la cada celda (gracias a la función "GenerateMap", "GenerateWorld" y "GeneratePriceMap"). Seguidamente, se procede a rellenar el mapa con las casillas sin descubrir y las descubiertas ("PaintOverMap" para las no descubiertas y "PaintMap" para las descubiertas). A continuación, se ejecuta la función de "PutMainSpecieDown", que añade la especie del jugador, junto con las secundarias. Luego se delimita el alcance de la cámara con la función "StartUp". Además, se inician los bucles con las funciones de "Feeding" (dedicada a calcular la población de todas las especies cada segundo), y la de "Catastrofes" (dedicada a la aparición de catástrofes cada pocos minutos). Finalmente, se pausa la ejecución del juego para que el jugador pueda leer el tutorial del juego.

```
private void Start()
{
    cellManager.CreateMap(mapGenerator.GenerateWorld(), mapGenerator.GeneratePriceMap());
    mapManager.PaintOverMap(mapGenerator.size);
    mapManager.PaintMap(CellManager.CellsDisc);

    PutMainSpecieDown();
    Vector3Int limits = new Vector3Int(mapGenerator.size-1, mapGenerator.size-1, 0);
    cameraController.StartUp(mapManager.GetMainMap().CellToWorld(limits).x,
        mapManager.GetMainMap().CellToWorld(limits).y);
    InvokeRepeating("Feeding", 1, 1);
    InvokeRepeating("Catastrofes", 300, 240);
    Time.timeScale = 0f;
}
```

Figura 22: Función "Start" de GameManager.

4.3.2 MapManager

En esta clase se guardan los distintos mapas que ve el jugador, específicamente 3 mapas:

1. El mapa principal que es el de los biomas.
2. El mapa de las casillas ocultas.
3. El mapa con las casillas premio.

Estos mapas son de la clase “Tilemap”, y se pueden añadir objetos del tipo “Tile” a una de sus coordenadas (del tipo Vector3Int). Para poder manejar más cómodamente estos mapas, se han creado funciones bastante útiles como “PaintMap”, que imprime todas las casillas que estén dentro de la variable “view”, llamando a la función “PaintTile”.

También se guardan otras variables, como el mapa desbloqueado por el jugador hasta ese momento (mediante una lista de Vector3Int llamada “view”). Gracias a la función “Addview” (Figura 23) que toma como argumento un Vector3Int, se añaden nuevas casillas a la lista y el jugador descubre nuevas partes del mapa. Esta función también hace referencia a otra dentro de la clase que se llama “CheckSpiralRing”, que calcula una espiral hexagonal rellena desde un punto concéntrico. Más adelante comentaré la dificultad de trabajar con un mapa hexagonal en vez de uno cuadrado.

```
public static void AddView(Vector3Int v)
{
    if (!view.Contains(v))
    {
        view.Add(v);
        PaintTile(v, CellManager.CellsDisc[v].tile);
        staticOver.SetTile(v, null);
    }
    foreach (Vector2Int auxV in CheckSpiralRing(height,new Vector2Int(v.x,v.y),20))
    {
        if (!view.Contains(new Vector3Int(auxV.x, auxV.y,0)))
        {
            view.Add(new Vector3Int(auxV.x, auxV.y, 0));
            staticOver.SetTile(new Vector3Int(auxV.x, auxV.y, 0), null);
            PaintTile(new Vector3Int(auxV.x, auxV.y, 0), CellManager.GetCell(new Vector3Int(auxV.x, auxV.y, 0)).tile);
        }
    }
}
```

Figura 23: Función “AddView”.

4.3.3 CellManager

En esta clase se guarda un diccionario con todas las celdas del mapa y su información. El diccionario está creado mediante una clave-valor de un Vector3Int y una HexCell (Dictionary<Vector3Int, Hexcell> cellsDic), ya que la clase Tilemaps Unity guarda las coordenadas como Vector3Int, y la clase Hexcell se guarda la información de la celda.

La información de cada celda se almacena en objetos de la clase HexCell (Figura 24). Esta clase contiene variables que debe mostrar como la Tile, sus coordenadas, su bioma actual e inicial, si contiene alguna especie y algunos datos relevantes para el gameplay.

```
public class HexCell
{
    public Tile tile;
    public Vector3Int coordinates;
    public BiomaType bioma;
    public BiomaType initialBioma;
    public int food;
    public Specie specie = null;
    public bool hasPrice = true;
    public AlimentosType alimentos;
    public ClimateType climate;
```

Figura 24: Variables de HexCell.

Dentro de CellManager contamos con diferentes funciones que son utilizadas repetidamente a lo largo del proyecto. Una de ellas es “GetNeigh”, que recibe como argumento un Vector3Int y devuelve la información de las celdas vecinas como una lista de HexCell. El reto ha sido conseguir los vecinos de una celda hexagonal, ya que, o bien almacenas los datos de una forma específica, o desarrollas funciones que faciliten la localización de las celdas (Figura 25). En este juego he optado por utilizar un almacenamiento de las coordenadas Offset, como si fuera un array bidimensional, con 2 variables de posición (x e y). Y he programado una función para pasar de coordenadas Offset a coordenadas cúbicas y así poder trabajar con ellas, después, tras conseguir los valores deseados, vuelven a pasar a Offset.

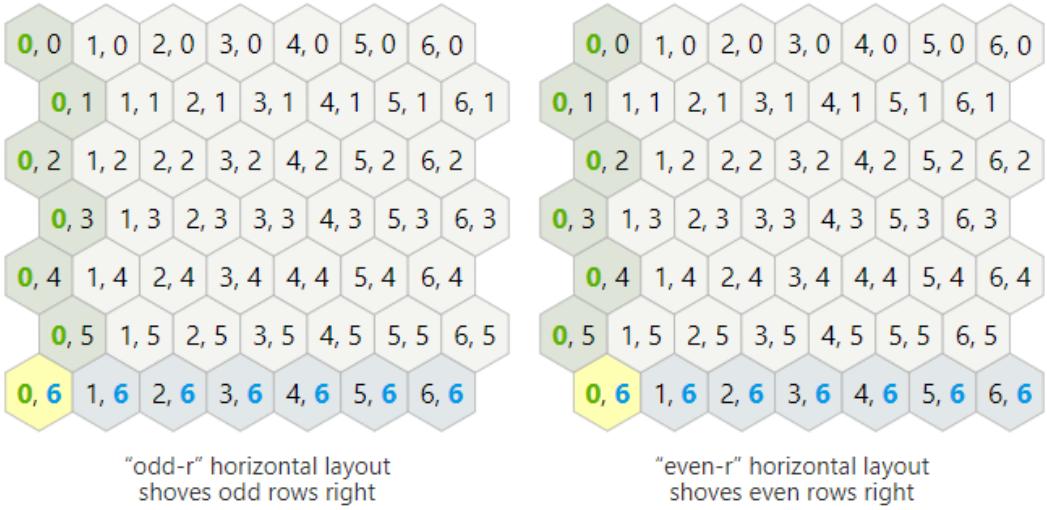

Figura 25: Mismas coordenadas, diferentes vecinos.

4.3.4 MapGenerator

Como bien indica el nombre de esta clase principalmente genera el mapa de juego. Al ser uno de los objetivos del trabajo poder explicar correctamente los biomas, se ha intentado que estos guarden el mayor parecido con los biomas actuales (Figura 26), por lo que se ha utilizado un método avanzado y complejo para poder generar mapas de estas características.

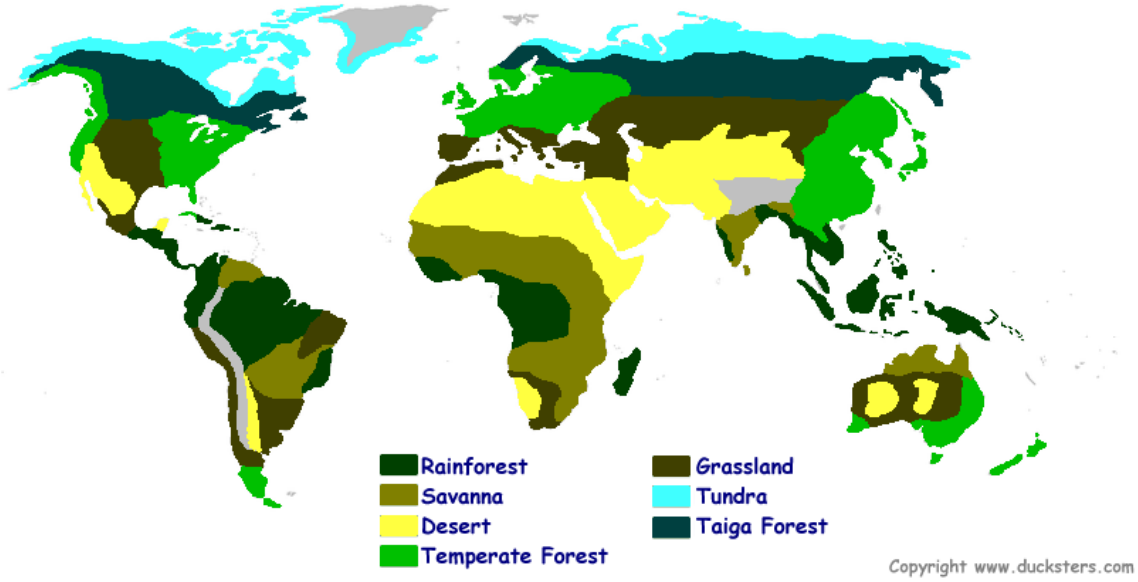

Figura 26: Biomas del mundo.

Para poder conseguir generar mapas orgánicos, he utilizado un método de expansión de un array bidimensional, que comienza con casillas de tierra y mar y al final se acaban incorporando el resto de biomas. La lógica que sigue es la siguiente: se empieza con un array pequeño (por ejemplo, de 2x2) y se va haciendo un zoom mediante el cual casi duplicamos el tamaño del mismo (N -> 2*N-1). Por cada vecino 2x2 del array original, producimos un array de 3x3, en el que las esquinas mantienen los valores originales, y el resto de valores se escoge en base a sus vecinos (Figura 27):

a	b
c	d

--- zooom -->

a	a or b	b
a or c	a or b or c or d	b or d
c	c or d	d

Figura 27: Zoom del array inicial.

Este patrón se repite para arrays más grandes (los valores resaltados son los que se han mantenido del array original) (Figura 28):

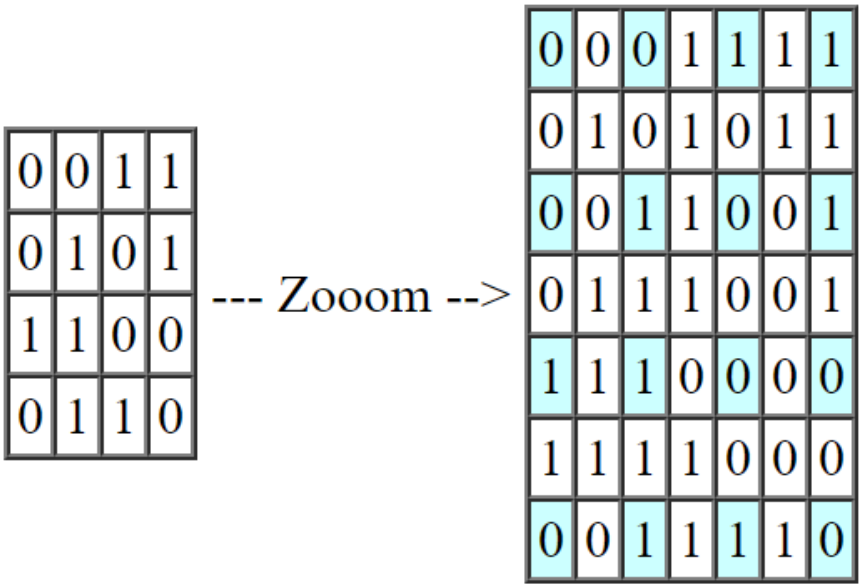

Figura 28: Zoom de 4x4 a 7x7.

La idea principal es que hay una imagen de baja resolución a la que hacemos zoom, en cada iteración del zoom le vamos añadiendo detalles aleatorios (los números aleatorios). Un ejemplo de cómo funciona este método con diferentes números (o biomas) es el de la figura 29.

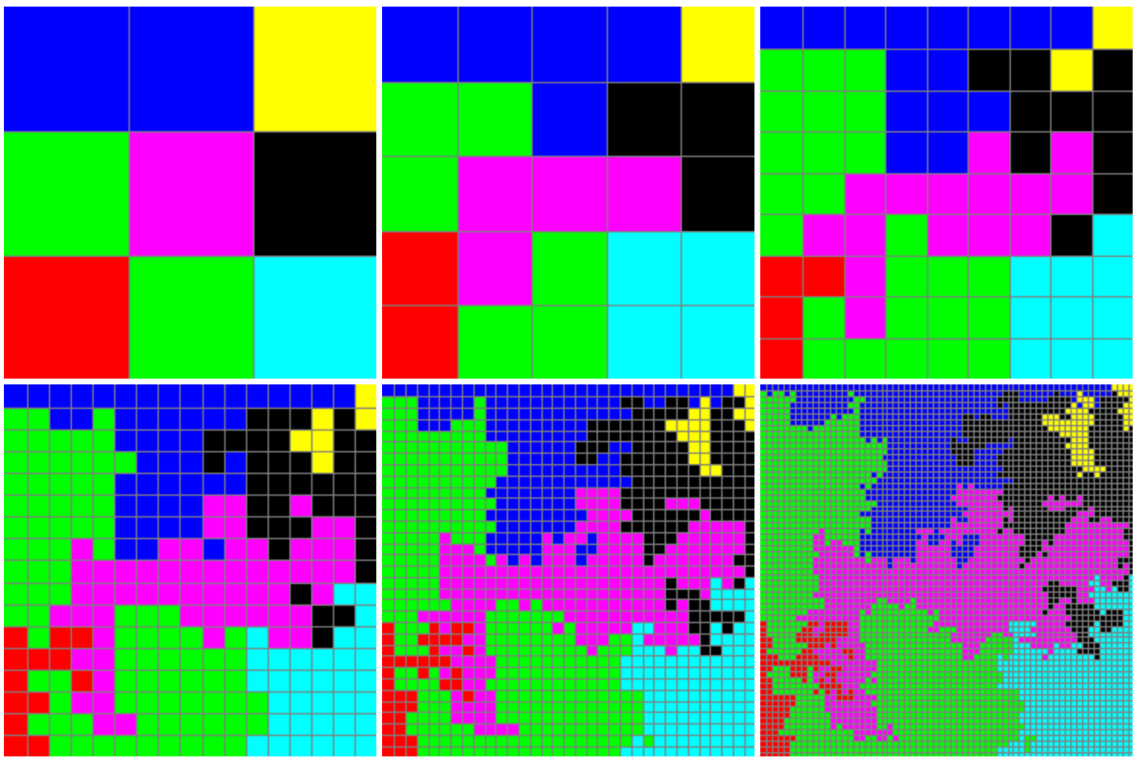

Figura 29: Ejemplo de generación de mapa.

Una vez explicado cómo funciona la generación de biomas, voy a explicar detalladamente el siguiente fragmento de código (Figura 30):

```
public int[,] GenerateWorld()
{
    int[,] continents = GenerateContinents(9);
    continents = ResampleWorld(continents, 17);
    size = 17;
    world = FillFirstArray(continents, size);
    RepeatWorld(3);
    world = DeleteAlone(world);
    world = AddDeepOceanAndOthers(world);
    MapManager.width = size;
    MapManager.height = size;
    return world;
}
```

Figura 30: Función GenerateWorld de MapGenerator.

Como he indicado antes, la fórmula de crecimiento del array es [N -> 2*N-1], por lo que deberemos trabajar con números específicos para que funcione nuestro algoritmo. Dichos números son: 3, 5, 9, 17, 33, 65, 129, 257… He decidido que 129 es un tamaño suficientemente alto para representar los biomas y que haya suficientes especies por todo el mapa.

La primera línea de la función que llama a “GenerateContinents(9) inicializa un array de 9x9 con la mitad de casillas terrestres y la otra mitad marinas. Al ir a colocar una casilla terrestre, se comprueba que no tenga casillas terrestres a menos de 2 casillas, en tal caso, se coloca junto a una de éstas para así evitar crear un mapa lleno de islas que no conectan entre sí, y genera un continente medianamente grande.

Seguidamente realizamos una iteración del algoritmo anterior mediante la función “ResampleWorld”, y nos devuelve un array de 17x 17, suficientemente grande para poder añadir los diferentes biomas terrestres y que no se desvíen mucho a medida que se hace zoom. Con la función “FillFirstArray” se consegue cambiar los trozos de tierra por biomas predefinidos (Figura 31) en base a la altura “y” dónde estén en el mapa. La función “CoinFlip” devuelve un valor numérico de los biomas que le hemos pasado como argumento y la probabilidad con la que pueden aparecer. Los biomas terrestres con los que se trabaja en esta función son los siguientes: Tundra = 2, Bosque = 3, Jungla = 4, Sabana = 5, Pradera = 6, Desierto = 7, Ártico = 8. También añadimos el bioma 11 (Helada) en los polos.

```
if (continents[x, y] == 3)
{
    switch (y)
    {
        case 0:
        case 16:
            newWorld[x, y] = 8;
            break;
        case 1:
        case 15:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(8, 0.75f), new Vector2(2, 0.25f) });
            break;
        case 2:
        case 14:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(8, 0.25f), new Vector2(2, 0.75f) });
            break;
        case 3:
        case 13:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(2, 0.50f), new Vector2(3, 0.50f) });
            break;
        case 4:
        case 12:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(3, 0.75f), new Vector2(6, 0.25f) });
            break;
        case 5:
        case 11:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(3, 0.50f), new Vector2(6, 0.50f) });
            break;
        case 6:
        case 10:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(6, 0.50f), new Vector2(5, 0.40f), new Vector2(4, 0.10f) });
            break;
        case 7:
        case 9:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(6, 0.35f), new Vector2(5, 0.40f), new Vector2(4, 0.25f) });
            break;
        case 8:
            newWorld[x, y] = CoinFlip(new List<Vector2> { new Vector2(7, 0.50f), new Vector2(5, 0.35f), new Vector2(4, 0.15f) });
            break;
        default:
            break;
    }
}
else
{
    newWorld[x, y] = 1;
    //Añadimos las heladas
    if (y<2||y>14)
        newWorld[x, y] = 11;
}
```

Figura 31: Parte de la función FillFirstArray.

Posteriormente, se llama a la función “Repeat World” con el número de veces que queremos que itere el algoritmo explicado anteriormente (en este caso 3), por lo que convertirá nuestro mapa de tamaño 17x17 a uno de 129x129. Además, se eliminan las casillas que se han quedado aisladas en un bioma que no les corresponde (por ejemplo, una casilla de bosque rodeada por los 6 lados por desierto).

Por último, con la función de “AddDeepOceanAndOthers” (Figura 32) se añade el océano profundo. Inicialmente el mapa estaba cubierto por océano poco profundo, y se cambian todas las casillas de océano poco profundo que no se encuentren a una distancia menor o igual que 2 de un bioma terrestre. Una vez añadido el océano poco profundo, se eliminan los océanos que sean demasiado pequeños. Para esto utilizamos la función “Bucket”, que nos devuelve una lista de listas de Vector2Int. Cada lista almacena las casillas del bioma que se quiere consultar y, en este caso, si el tamaño del bioma de océano profundo es menor que 100, se cambian sus casillas por océano poco profundo. Seguidamente se añaden los biomas de abismo con la función “AddAbism”, que coloca casillas de abismo en el bioma de océano profundo que esté a cierta distancia de la tierra (si hay otro abismo en cierto rango, se coloca justo a su lado). Y finalmente se añaden los remolinos, que empiezan en una casilla del océano profundo y se colocan en una dirección. Esta dirección puede cambiar de forma

aleatoria, y el remolino se detiene si en la siguiente casilla que se va a colocar es océano poco profundo, abismo u otro remolino.

```
private int[,] AddDeepOceanAndOthers(int[,] map)
{

    //Añadimos el oceano profundo
    int arSize = map.GetLength(0);
    int[] biomes = { 1, 9 };
    for (int x = 0; x < arSize; x++)
    {
        for (int y = 0; y < arSize; y++)
        {
            if (map[x, y] == 1)
            {
                if (CheckRingBiomas(map, new Vector2Int(x, y), 1, biomasTerrestres))
                    continue;
                if (CheckRingBiomas(map, new Vector2Int(x, y), 2, biomasTerrestres))
                    continue;
                map[x, y] = 9;
            }
        }
    }

    List<List<Vector2Int>> lista = Bucket(map, new List<int>{9});
    List<List<Vector2Int>> auxLsita = new List<List<Vector2Int>>(lista);
    foreach (List<Vector2Int> l in auxLsita)
    {
        if (l.Count <= 100)
        {
            foreach (Vector2Int v in l)
            {
                map[v.x, v.y] = 1;
            }
            lista.Remove(l);
        }
    }

    //Añadimos los fondos oceánicos
    map = AddAbism(map, lista);

    //Añadimos los remolinos
    map = AddWhirlpool(map, lista);

    return map;
}
```

Figura 32: AddDeepOceanAndOthers.

Una vez realizadas estas funciones, se crea un array bidimensional lleno de números que representan los diferentes biomas anteriores. Una vez el array se muestra por pantalla y se le asigna a cada casilla su Tile correspondiente, el resultado es semejante a las Figuras 33 y 34:

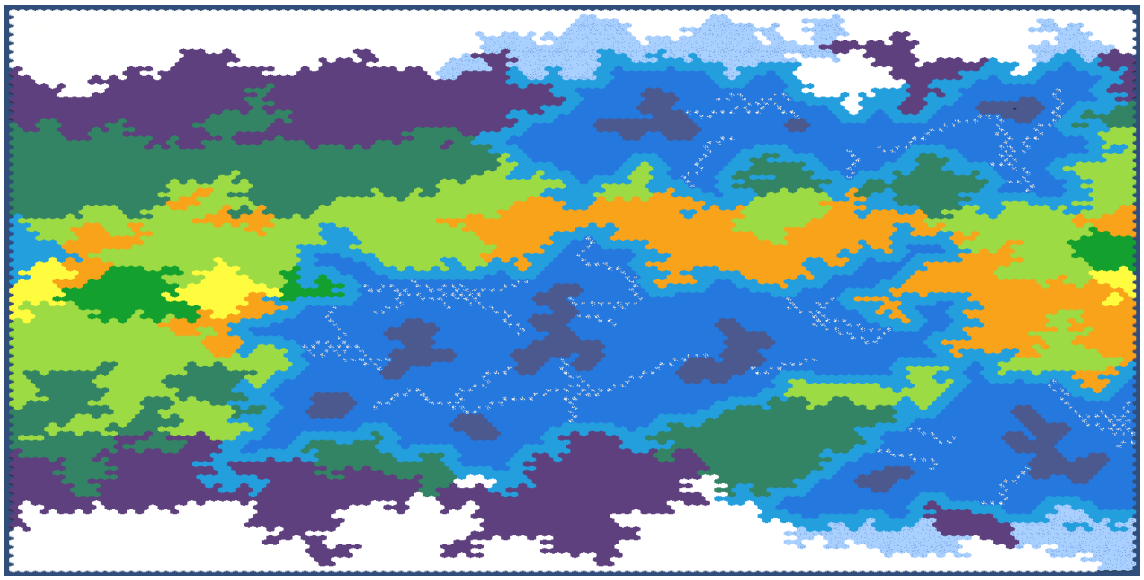

Figura 33: Mapa generado 1.

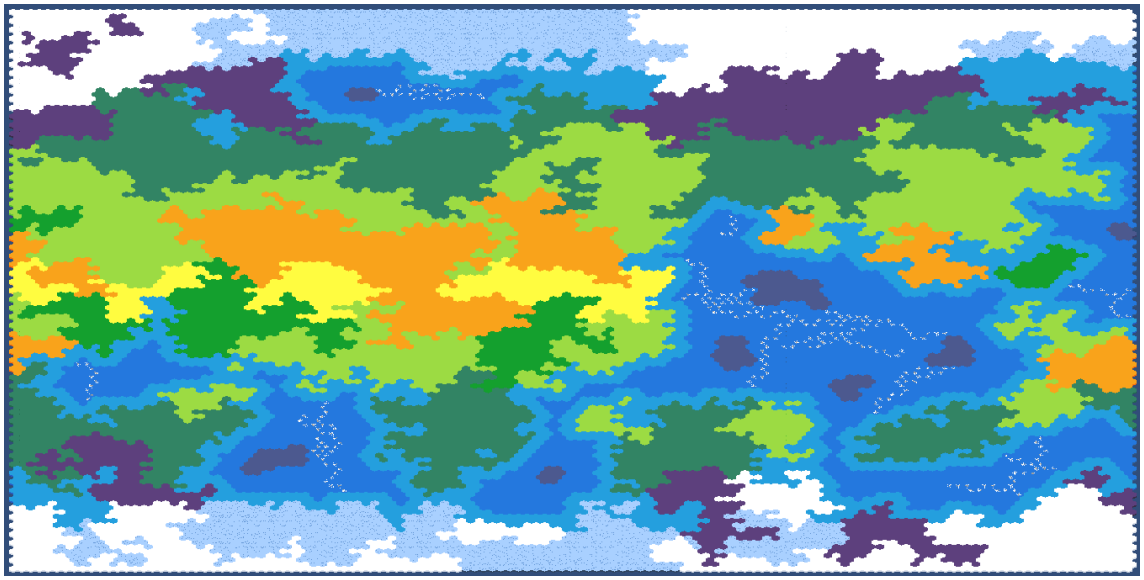

Figura 34: Mapa generado 2.

4.3.5 SpeciesManager

En esta clase se gestionan y se almacenan las especies que aparecen en el mapa. Un ejemplo de gestión de especies es la función "AddNewSpecie" que, si una especie no existe en el mapa, la añade a la lista de especies y la posiciona en el mapa. También se encuentra una de las funciones más importantes de todo el proyecto: la función de "Feeding" (Figura 35). Como se ha indicado en la clase GameManager, esta función se ejecuta cada segundo, y simula la expansión natural que tiene una especie en un medio.

```
public void Feeding()
{
    foreach (Specie specie in speciesList)
    {
        Dictionary<Vector3Int, SpecieCell> auxSpecieCells = new Dictionary<Vector3Int, SpecieCell>(specie.specieCells);
        foreach (KeyValuePair<Vector3Int, SpecieCell> entry in auxSpecieCells)
        {
            HexCell tile = CellManager.GetCell(entry.Key);
            float specieFood = tile.food;
            int newPop=0;
            if (specie.HowSuitable(tile) > 0)
                newPop = FeedingFormula(specieFood, specie, entry.Value.population,0);
            else
            {
                    newPop = FeedingFormula(specieFood, specie, entry.Value.population, -0.3f);
            }
            if (newPop >= 2)
            {
                specie.ModifyPopulation(tile.coordinates, newPop);
            }
            else
            {
                specie.RemoveSpecieFromOldCell(tile.coordinates);
            }
        }

    }
}
```

Figura 35: Función Feeding.

Cada iteración dentro del segundo bucle foreach calcula la nueva población de la casilla que alberga a la especie mediante la función "FeedingFormula". Esta función está inspirada en un modelo matemático de tipo logístico para dinámicas poblacionales, que varía en función de los parámetros (Figura 36).

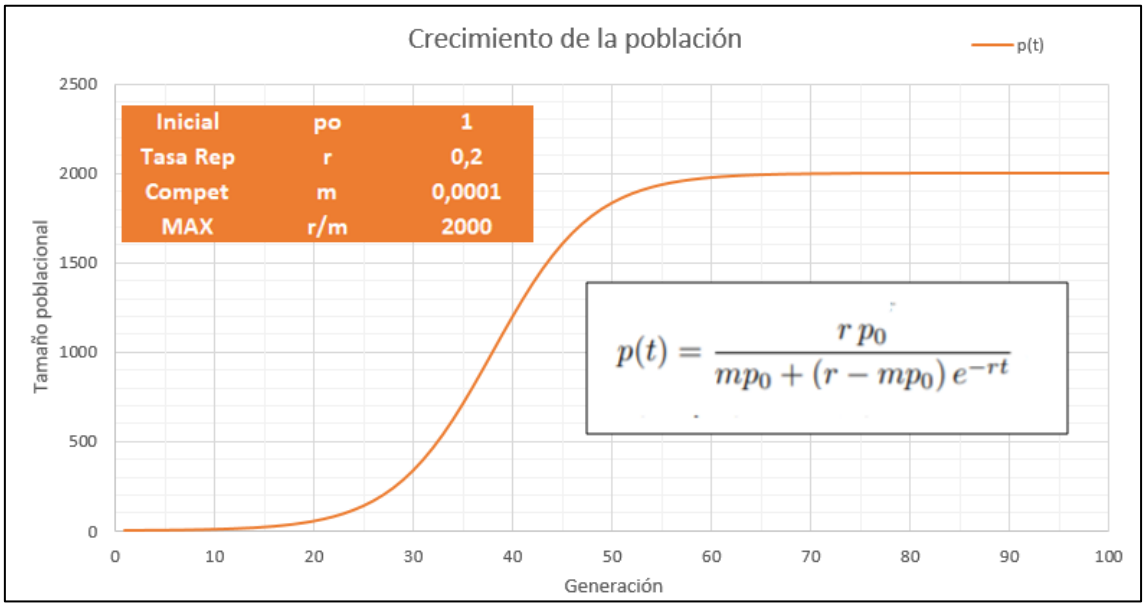

Figura 36: Dinámicas de poblaciones.

Finalmente existen dos funciones que añaden especies secundarias al mapa: "AddEspeciesSecundarias" y "ReponerEspecies". La función "AddEspeciesSecundarias" se encarga de añadir las especies secundarias al comienzo de la partida, y la de "ReponerEspecies" se encarga de añadir las especies que han desaparecido tras una catástrofe. Ambas funciones toman el bioma preferido de la

especie que van a colocar, y buscan los biomas más grandes que hay en el mapa para colocarlas.

4.3.6 Catastrophe

Esta clase es la encargada de producir y manejar las catástrofes que ocurren en el juego y sus consecuencias. Cada cierto tiempo se llama a la función "StartCatastrophe" (Figura 37) desde la clase "GameManager", esto ejecuta las corutinas de las diferentes catástrofes disponibles.

```
public void StartCatstrophe()
{
    soundManager.PlayCatastrofe();
    switch (CatastrofeCount%5)
    {
        case 0:
            StartCoroutine(Incendio());
            break;
        case 1:
            StartCoroutine(BajadaTemperaturas());
            break;
        case 2:
            StartCoroutine(SubidaTemperaturas());
            break;
        case 3:
            StartCoroutine(SubidaMar());
            break;
        case 4:
            StartCoroutine(Meteoro());
            break;
        default:
            break;
    }
    CatastrofeCount++;
}
```

Figura 37: Función StartCatastrophe.

Cada una de las catástrofes afecta de manera diferente al mapa:

-**Incendio:** el incendio comienza en una casilla de la superficie terrestre más grande del mapa, y se va expandiendo por el mapa cada 0.2 segundos, dejando a su paso tierra infértil en la que las especies no pueden habitar.

-**Bajada de Temperaturas:** la bajada de temperaturas primero congela los biomas marinos llegando al ecuador del mapa, pero dejando un pequeño espacio sin congelar. Seguidamente congela los terrestres (convierte cada bioma en uno más frío, por ejemplo: Tundra>Ártico, Pradera>Tundra, Jungla>Bosque).

-**Subida de Temperaturas:** la subida de temperaturas primero seca los biomas marinos casi por completo dejando a su paso desierto, y seguidamente aumenta la temperatura de los terrestres (convierte cada bioma en uno más cálido, por ejemplo: Sabana>Desierto, Pradera>Sabana, Ártico>Bosque).

-**Subida del Mar:** la subida del mar invade los biomas terrestres con biomas marinos, pero sin cubrirlos por completo.

-**Meteoro:** el meteoro sacude la pantalla del jugador justo antes de provocar un pequeño incendio y la subida de los mares en el resto del mapa.

4.3.7 CameraController

La clase de CameraController gestiona el movimiento de la cámara. El jugador puede moverse por el mapa cómodamente gracias a esta función. Las funciones principales que cumple son: mover la cámara en la dirección que el usuario desee y acercar/alejar el zoom. Para mover la cámara (Figura 38), el usuario puede moverse con las teclas awsd o llevando el ratón a los bordes de la pantalla (limitado por la variable "panBorderThickness"). Una vez se realizan alguna de estas acciones, se calcula el tiempo que ha pasado y se multiplica por la velocidad de movimiento que le hemos asignado a la cámara ("panSpeed"). Para el zoom, se comprueba que la rueda del ratón se haya movido, y en base a eso se aumenta o disminuye el tamaño de la cámara (la clase cuenta con dos funciones que permiten cambiar el zoom desde los botones del minimapa de la Interfaz de Juego).

```
void Update()
{
    //Evita que pulsemos sobre Elementos de la interfaz

    time = Time.realtimeSinceStartup - oldTime;
    Vector3 pos = transform.position;
    float auxSize = this.GetComponent<Camera>().orthographicSize;

    bool canmove = !(UIManager.specieMenuActive || UIManager.pauseMenuActive);
    bool overUI = !EventSystem.current.IsPointerOverGameObject();
    if (canmove &&(Input.GetKey("w") || (Input.mousePosition.y >= Screen.height - panBorderThickness && overUI)))
    {
        pos.y += panSpeed * time;
    }
    if (canmove && (Input.GetKey("s") || (Input.mousePosition.y <= panBorderThickness && overUI)))
    {
        pos.y -= panSpeed * time;
    }
    if (canmove && (Input.GetKey("d") || (Input.mousePosition.x >= Screen.width - panBorderThickness && overUI)))
    {
        pos.x += panSpeed * time;
    }
    if (canmove && (Input.GetKey("a") || (Input.mousePosition.x <= panBorderThickness && overUI)))
    {
        pos.x -= panSpeed * time;
    }
    pos.x = Mathf.Clamp(pos.x, 0, panLimit.x);
    pos.y = Mathf.Clamp(pos.y, 0, panLimit.y);
    transform.position = pos;
```

Figura 38: Update de CameraController.

4.3.8 Specie

La clase Specie almacena la información de la especie que le corresponda y posee diferentes funciones para que la especie funcione correctamente en el juego. Se compone de 21 variables que almacenan datos como el nombre, los rasgos evolutivos, las celdas en las que está expandido, la población total, etc. Las funciones están estructuradas en diferentes secciones del código dependiendo del uso que tengan.

4.3.8.1 Modificadores de población

En esta categoría se encuentran 3 funciones cuya funcionalidad está relacionada con modificar la población de la especie de diferentes maneras, ya sea modificando la población de una celda existente, añadiéndola a una nueva celda o eliminándola de una antigua celda.

La función "ModifyPopulation" modifica el número de individuos de una celda existente. Primero se comprueba si la especie cumple con los requisitos para poder expandirse en el mapa y así cambiar el valor de la celda. Si la población es positiva, se añade la diferencia y se actualiza el valor del diccionario. Si el valor es negativo, se llama a la función "RemoveSpecieFromOldCell", la cual elimina tanto la población como la especie de una casilla concreta.

La función "AddSpecieToNewCell" (Figura 39) añade nuestra especie a una nueva casilla en la que no estaba expandida antes. Para realizar esto es necesario seguir una serie de medidas para que funcione correctamente. Primero, se añade la especie a la variable specie de la "HexCell" correspondiente (como se comenta en el punto 4.3.3). Seguidamente se añade la población nueva a la población total y se añade una nueva "SpecieCell" al diccionario de celdas de la especie (la clase "SpecieCell" alberga información útil de la celda como las coordenadas, la población de dicha celda, si se puede expandir, etc). Luego se llama a la función "Checkfrontier" que actualiza la lista con las celdas que colindan con celdas que no poseen la especie actual. Después coloreamos la celda del color de nuestra especie y actualizamos el Tilemap para ocupar el menos espacio posible. Finalmente añadimos los bordes a la nueva casilla gracias a la clase border, que coloca los bordes de la especie en el mapa.

```
public virtual void AddSpecieToNewCell(Vector3Int coordinates)
{
    CellManager.AddSpecieToCell(coordinates, this);
    totalPopulation += startingMembers;
    AddSpecieCell(coordinates);
    CheckFrontier(coordinates);
    specieMap.SetTile(coordinates, specieTile);
    specieMap.CompressBounds();
    border.CheckBorders(coordinates, this);
}
```

Figura 39: AddSpecieToNewCell.

La función "RemoveSpecieFromOldCell" (Figura 40) realiza funciones similares a las de "AddSpecieToNewCell" pero su función es eliminar a la especie de una celda del mapa. Primero se elimina la celda que se había añadido al Tilemap. Seguidamente se elimina la celda de la casilla HexCell del CellManager. Luego se resta la población que tenía esta celda de la población total. Después se elimina de nuestro diccionario de celdas de la especie "SpecieCells", y finalmente se actualiza la frontera con "CheckFrontier".

```
public virtual void RemoveSpecieFromOldCell(Vector3Int coordinates)
{
        specieMap.SetTile(coordinates, null);
        specieMap.CompressBounds();
        CellManager.RemoveSpecieFromCell(coordinates, this);
        totalPopulation += -specieCells[coordinates].population;
        specieCells[coordinates].DeleteSpecieCell();
        specieCells.Remove(coordinates);
        CheckFrontier(coordinates);
        border.CheckBorders(coordinates, this);
}
```

Figura 40: RemoveSpecieFromOldCell.

4.3.8.2 Expansión

En esta categoría se encuentran diferentes funciones que son utilizadas para la expansión de la especie a nuevas casillas o para la comprobación de la posibilidad de expandirse a una nueva celda. Un concepto importante de este apartado es la variable "frontier", que es una lista de las celdas que chocan con los bordes de las celdas expandidas y a las que nos podemos expandir bajo ciertas circunstancias. He utilizado este método para mejorar la eficiencia del juego ya que, si al hacer los cálculos tuviéramos que calcular la frontera cada vez que cada especie la necesita, implicaría un tiempo de cálculo muy largo.

La función "ExpansionLoop" (Figura 41) se ejecuta continuamente en la función de Update. Cada cierto tiempo (expansionTime) la especie se intenta expandir a una nueva casilla (mediante el método "Expand"). En el caso de no poder conseguirlo (porque todas las celdas de la frontera no son aptas), se mantendrá a la espera hasta que se actualice la lista de Frontera y se expandirá de inmediato cuando esto ocurra.

```
public void ExpansionLoop()
{
    if (expandTimer)
    {
        if (Expand())
        {
            expandTimer = false;
            Invoke("ExpandTimer", expansionTime);
        }
    }
}
public void ExpandTimer()
{
    expandTimer = true;
}
```

Figura 41: ExpansionLoop.

La función "Expand" (Figura 42) recorre todas las celdas de frontier y calcula si tiene una celda colindante de la especie que se pueda expandir. En tal caso, calcula el nivel de adaptación que tiene en esa celda mediante la función "ExpansionSuitable". Este se guarda en una lista con su coordenada y su nivel de

adaptación, y posteriormente se ordena y se elige la coordenada con una puntuación mayor.

```
public virtual bool Expand()
{

    List<ExpansionTuple> listHex = new List<ExpansionTuple>();
    HexCell expansionCell = new HexCell();
    foreach (Vector3Int cell in frontier)
    {
        if (!AvailableExpansion(CellManager.GetCell(cell)))
            continue;

        bool markExp = false;
        foreach (HexCell auxCell in CellManager.GetNeigh(CellManager.GetCell(cell).coordinates))
        {
            if (HasSpecie(auxCell.coordinates) && GetCanExpand(auxCell.coordinates))
            {
                markExp = true;
                break;
            }
        }
        if (!markExp) continue;
        float auxSuitValue = ExpansionSuitable(CellManager.GetCell(cell));
        if(auxSuitValue>-999)
            listHex.Add(new ExpansionTuple(auxSuitValue, CellManager.GetCell(cell)));
    }
```

Figura 42: Parte de la función Expand.

La función “CheckFrontier” (Figura 43) se ejecuta siempre al añadir/eliminar una celda de la especie, y comprueba si las celdas vecinas de una posición deben pertenecer a la frontera o no. Primero comprueba si la celda contiene la especie, en cuyo caso eliminaríamos la celda de la frontera. Y si no la contiene comprueba sus vecinos para confirmar si debe permanecer allí o no. Por último, comprueba todas las celdas vecinas para ver si alguna debe pertenecer a la frontera.

```
public void CheckFrontier(Vector3Int coordinates)
{
    if (HasSpecie(coordinates))
    {
        RemoveFromFrontier(coordinates);
    }
    else
    {
        if (HasSpecieNeigh(coordinates))
        {
            AddToFrontier(coordinates);
        }
        else
        {
            RemoveFromFrontier(coordinates);
        }
    }
    foreach (HexCell cell in CellManager.GetNeigh(coordinates))
    {
        if (HasSpecieNeigh(cell.coordinates)&&(!HasSpecie(cell.coordinates)))
        {
            AddToFrontier(cell.coordinates);
        }
        else
        {
            RemoveFromFrontier(cell.coordinates);
        }
    }
}
```

Figura 43: CheckFrontier.

5. EXPERIMENTOS

En este apartado se van a explicar las pruebas realizadas con los usuarios, los conocimientos que se adquieren con este videojuego y los cambios realizados para mejorar la usabilidad del juego. En total, 30 usuarios han probado el juego y han realizado 2 encuestas con las mismas preguntas: una antes de probar el juego (Pre-Test) y otra después de haberlo probado (Post-Test) durante 20 minutos. Se ha realizado de esta forma para determinar cuánto pueden llegar a aprender con el juego (aparte de reportar algún pequeño bug o proponer alguna mejora).

Al comienzo del cuestionario se realizan unas preguntas para poder separar y analizar las respuestas por género, edad y grado de conocimientos sobre la evolución biológica. El 70% de los participantes han sido hombres y el 30% mujeres. El 56.7% de los participantes tienen una edad comprendida entre los 12 y los 17 años, mientras que el 43.3% tienen una edad superior a 18 años. En cuanto a conocimientos sobre la evolución biológica, el 10% de los usuarios no tenían ningún conocimiento, el 33.3% tenían poco conocimiento sobre el tema, y el 56.7% tenían un conocimiento básico.

Todos los usuarios han conseguido una puntuación mayor tras haber utilizado el juego, pasando de una media de 39/63 puntos a una media de 54/63 puntos (Figura 44).

Figura 44: Puntuaciones antes y después de jugar.

El objetivo de la primera pregunta de la encuesta es comprobar si conocen algunos conceptos de la evolución biológica (Figura 45). Se plantean 3 afirmaciones erróneas sobre la evolución biológica y una cuarta opción que afirma que son falsas. En la primera encuesta realizada la mayoría de los usuarios escogen una opción incorrecta (solamente 3 aciertan), mientras que en la segunda encuesta 21 usuarios aciertan esta pregunta. Según estos datos, con el juego se consigue aprender sobre este concepto.

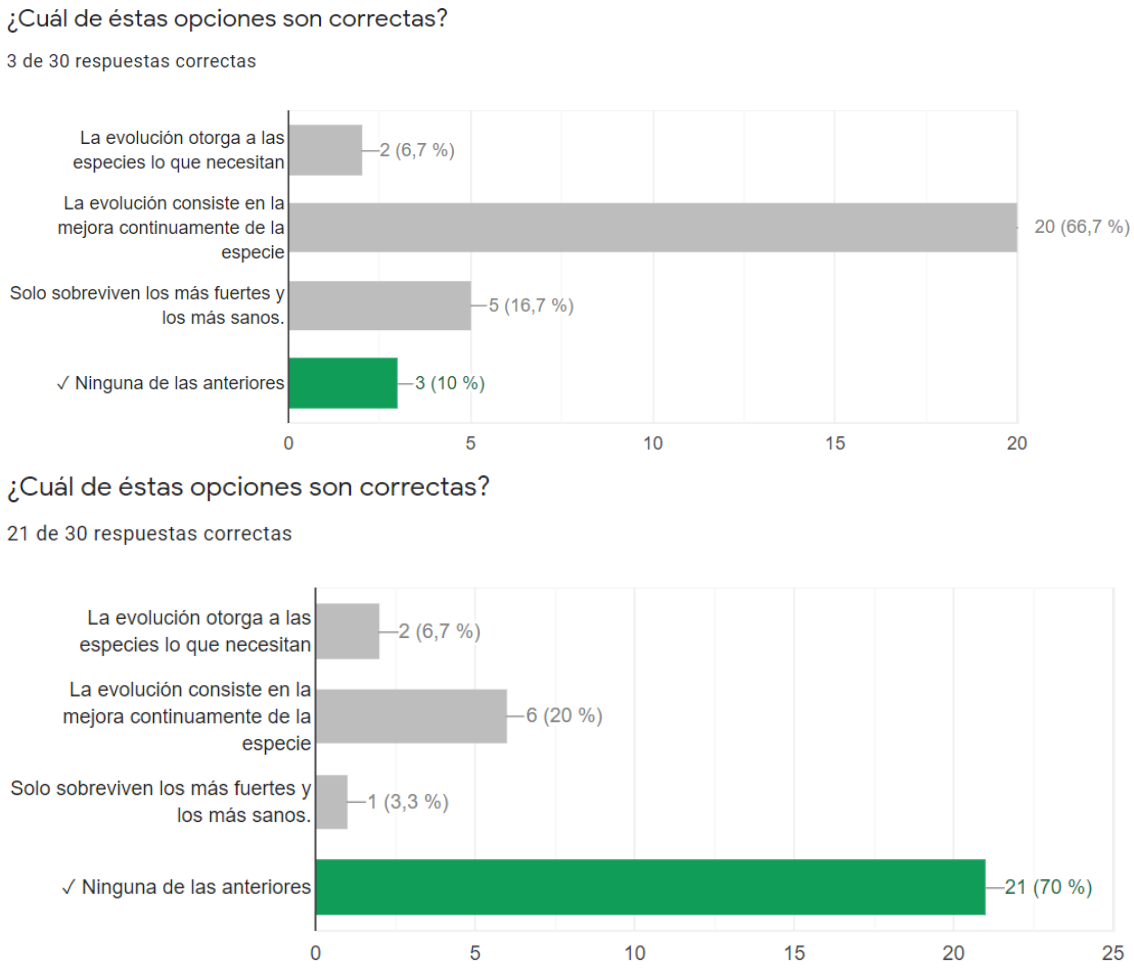

Figura 45: Pregunta 1.

Las preguntas 4 y 5 se centran en determinar qué grado de conocimiento tienen los participantes sobre rasgos evolutivos de las especies. Por ejemplo, en la pregunta 4 (Figura 46): “Si un animal posee los siguientes rasgos: Pelaje, Sangre caliente y Pulgar Opuesto, ¿a qué bioma está mejor adaptado?” 14 usuarios respondieron correctamente en el primer formulario, mientras que en el segundo 21 personas contestaron correctamente. Se consigue un 50% más de respuestas correctas en comparación con el formulario inicial.

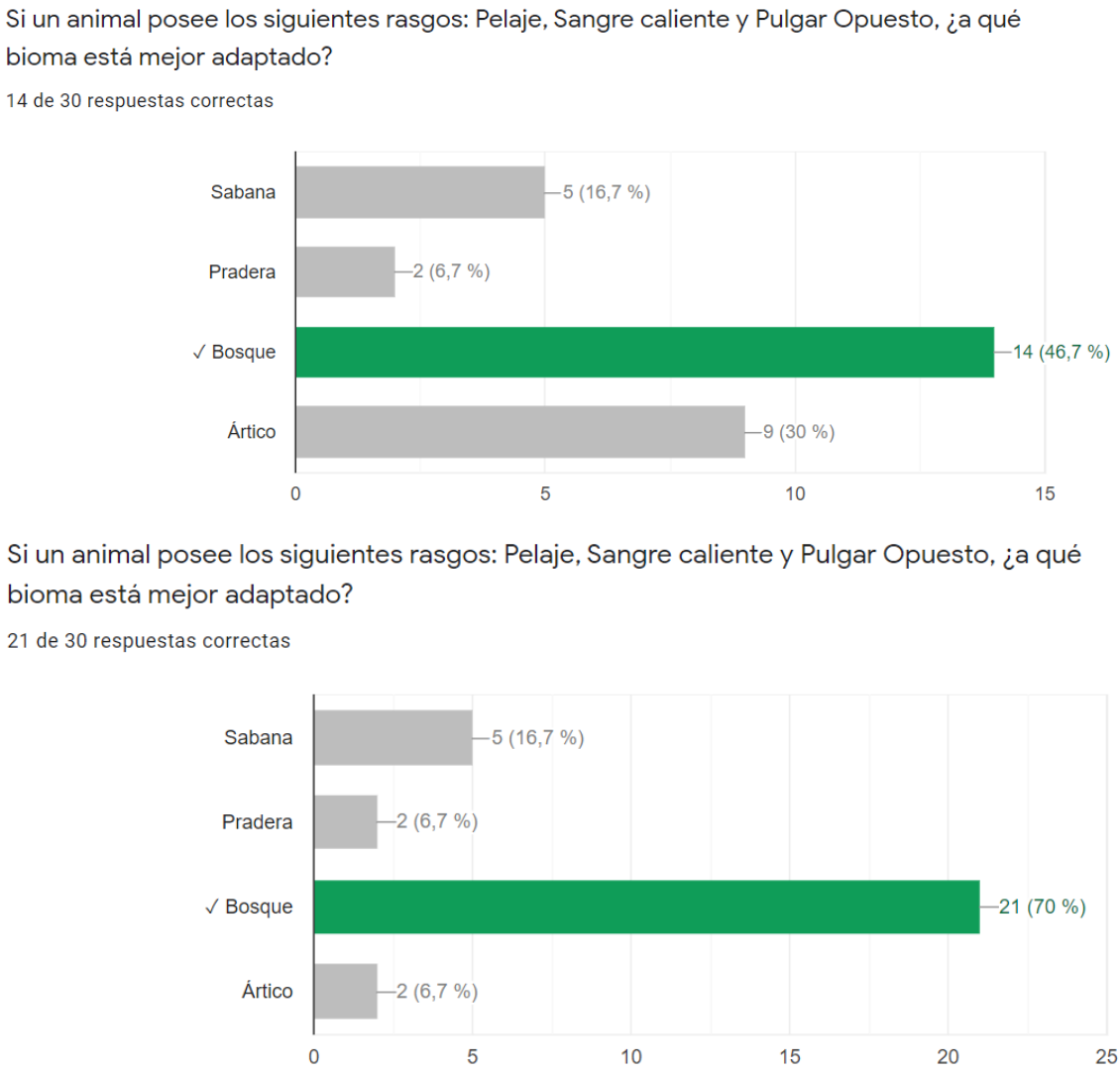

Figura 46: Pregunta 4.

Por último, una de las preguntas de las que más han aprendido los participantes ha sido la pregunta acerca de LUCA (Last Universal Common Ancestor en inglés, último antepasado común universal en español) (Figura 47). En la primera encuesta tan sólo 4 personas conocían este concepto, mientras que, en la segunda encuesta, 29 usuarios afirmaron haber comprendido el significado de este concepto.

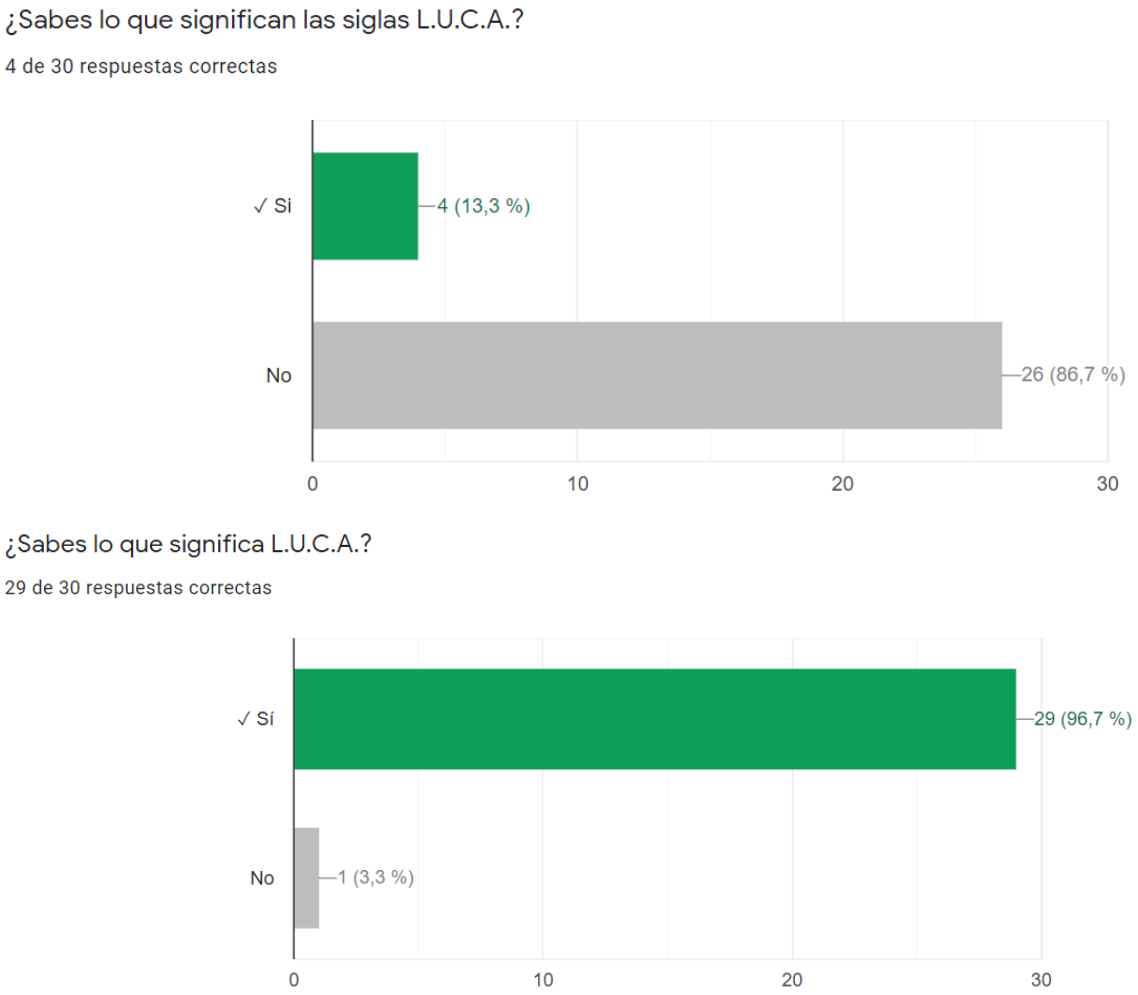

Figura 47: Pregunta 9.

Uno de los apartados de las encuestas está orientado a cómo mejorar el juego: si han encontrado algún bug, si les resultó confuso en algún momento, etc. En base a las respuestas de los usuarios, se han realizado los siguientes cambios:

-Para los usuarios de ordenador portátil sin ratón era bastante difícil controlar la cámara, por lo que se han añadido botones para hacer zoom en la interfaz, y cambiado el control para seleccionar la casilla a la que se quiere ir (antes era el botón central de la rueda del ratón, después el click derecho).

- Cuando termina la partida en vez de sacarte del juego se ha modificado para que el usuario vuelva al menú principal.

-Cuando aparecía una catástrofe, al cerrar la pestaña volvía a la velocidad normal aunque estuviera aumentada. este error se ha solucionado.

-Modificar los costes de las evoluciones de los rasgos, inicialmente eran muy bajos y los usuarios los conseguían extremadamente rápido.

-Que los usuarios no puedan evolucionar escamas y pelaje al mismo tiempo, así evitamos confusiones evolutivas, y mejora el gameplay.

-Una de las pantallas de tutorial no se cargaba correctamente cuando se volvía a él con la flecha de vuelta.

-La primera catástrofe ahora se produce a los 5 minutos, en vez de a los 2 minutos como inicialmente, ya que era demasiado pronto.

-Se han corregido algunas faltas de ortografía de los textos.

Los resultados observados en la Figura 48 y 50, sobre las cuales se ha realizado el t_test, muestra un valor p = 0.0001, muy por debajo del valor alfa = 0.05, por lo que se puede firmar que el aprendizaje de los conceptos teóricos ha sido significativo.

	Mínimo	Máximo	Media	Desv. Típica
PreTest	3,97	8,73	6,2113	1,1046
PostTest	5,71	10	8,6027	1,2446

Figura 48: Media y desviación típica del PreTest y PostTest.

	t test analysis	p-value
Pretest vs PostTest	-2,3913	0,0001

Figura 49: Estudio utilizando análisis t-student y p-value.

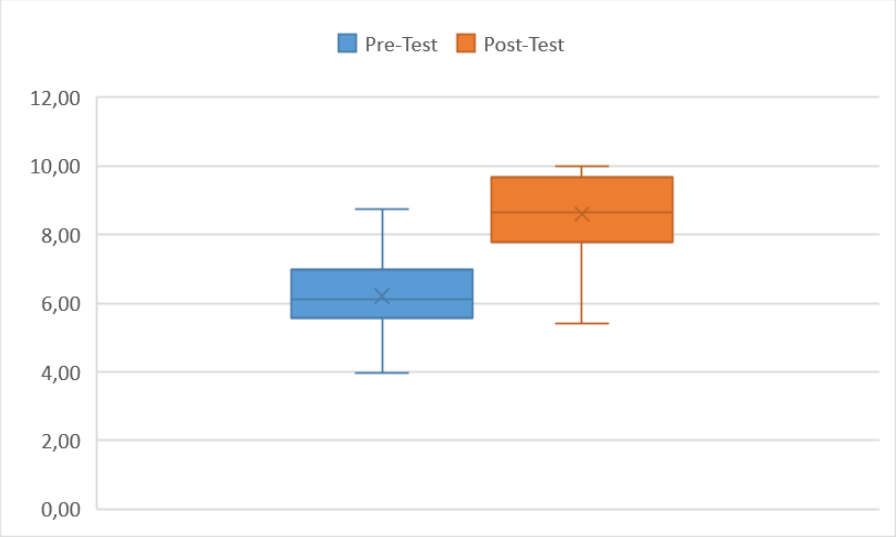

Figura 50: Boxplot del t-test.

6. CONCLUSIÓN

El objetivo del proyecto era crear un videojuego que enseñe conceptos básicos sobre la evolución biológica de forma transversal. En “L.U.C.A.” se simula la expansión de territorios de las especies con un modelo adaptativo flexible, algo no encontrado en los juegos ya existentes. Además, la mayoría de los juegos tratan la teoría evolutiva sin explicar la importancia de los rasgos que poseen las especies. Sin embargo, en “L.U.C.A.” los rasgos evolutivos de las especies son una característica relevante.

El juego se ha realizado con una estética estilo pixelart y está compuesto por una interfaz intuitiva y simple, que utiliza colores e iconos que el usuario ha utilizado en otras aplicaciones de este estilo. Se compone de dos escenas: la pantalla de inicio con el menú principal y la del juego. Lo más relevante desde el punto de vista de técnico es haber estructurado el código para que sea más eficiente y permita realizar procesos de múltiples especies simultáneamente consumiendo pocos recursos.

La aplicación conseguida permite aprender los conceptos básicos sobre la evolución biológica después de haberle dedicado un tiempo de juego mínimo de 20 minutos. Con las encuestas realizadas se ha podido comprobar que enseña los conceptos que se quieren enseñar. Podemos afirmar que hemos cumplido los objetivos principales del proyecto como explicar los biomas, los rasgos evolutivos, la selección natural y las catástrofes naturales.

7. BIBLIOGRAFÍA

Morten, M. (s. f.). *MortMort*. Google. https://www.mortmort.net/

Amit Patel, A. P. (s. f.). *Red Blob Games: Hexagonal Grids*. Red Blob Games. https://www.redblobgames.com/grids/hexagons/

Generating terrain in Cuberite. (s. f.). Cuberite. http://mc-server.xoft.cz/docs/Generator.html#preface

Technologies, U. (s. f.). *Unity - Manual: Unity User Manual (2018.4)*. Unity. https://docs.unity3d.com/2018.4/Documentation/Manual/

Technologies, U. (s. f.-b). *Unity - Scripting API:* Unity. https://docs.unity3d.com/ScriptReference/

Lospec - Free tools and resources for people making pixel art, voxel art and more. (s. f.). Lospec. https://lospec.com/

Intermediate Scripting. (s. f.). Unity Learn. https://learn.unity.com/project/scripting-de-nivel-intermedio

Beginner Scripting. (s. f.). Unity Learn. https://learn.unity.com/project/scripting-para-principiantes

Just a moment... (s. f.). Padlet. https://es.padlet.com/

Darwin, C. (2019). *El Origen de Las Especies: (spanish Edition)(Annotated) (Worldwide Classics)*. Independently Published.

Pertejo, A. J. M. (2016). *Biología. 2 Bachillerato. Savia* (1.ª ed.). EDICIONES SM.

Rodríguez, P. E. G. (2015). *Biología y geología, 1º Bachillerato, Savia*. SM.

Izura, A. P. X. (2019). *Introducción a Unity: Introducción al desarrollo de videojuegos con Unity 2D (Spanish Edition)*. Independently published.

Whitton, N. (2014). *Digital Games and Learning: Research and Theory*. Routledge.